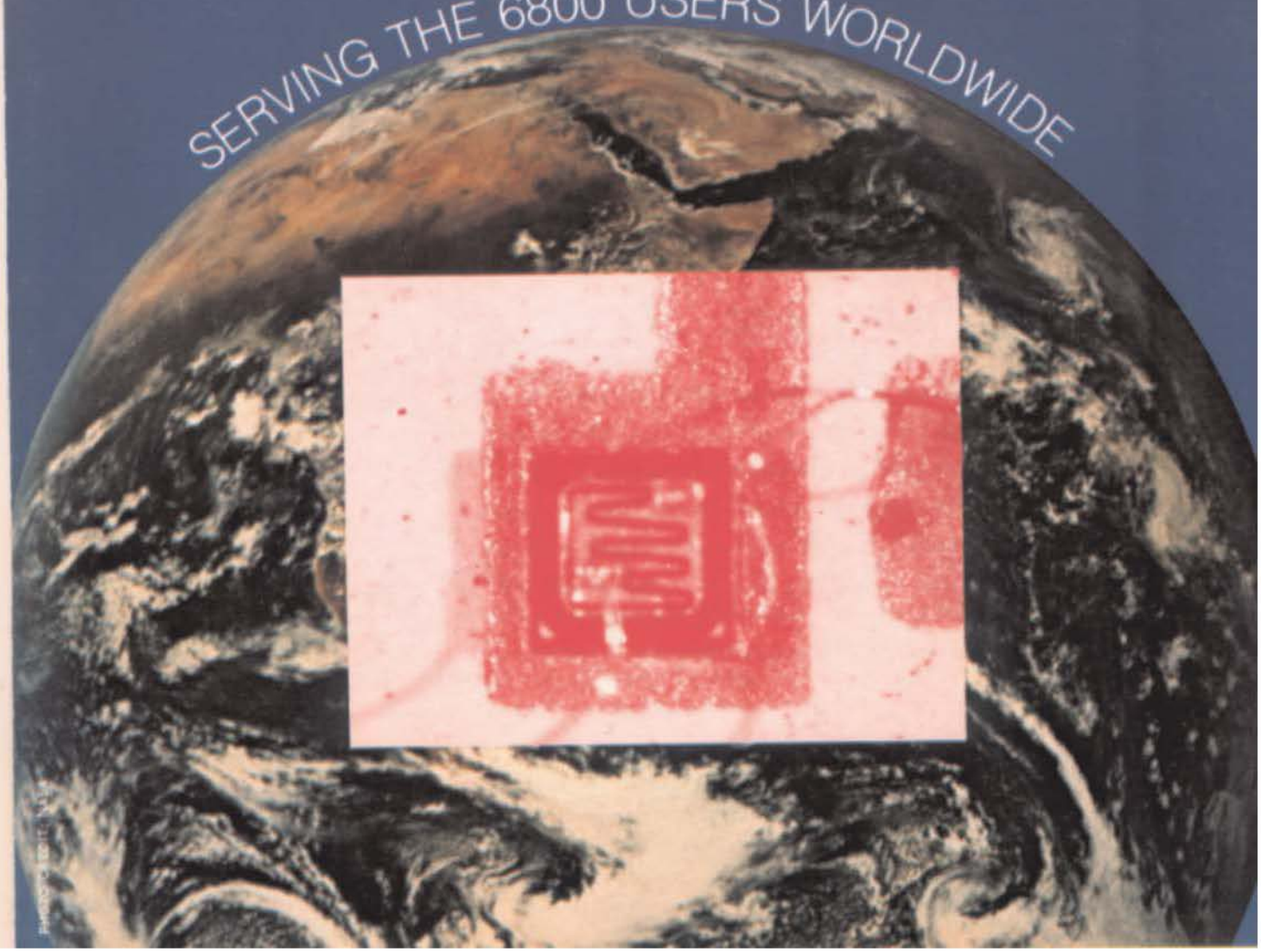
'68' MICRO JOURNAL

2.00

VOLUME I ISSUE 5 • Devoted to the 6800 User • July 1979

"Small Computers Doing Big Things."



SYSTEMS – SOLUTIONS

If you have a problem that can be solved by a computer—we have a systems solution.

- Two central processors with maximum RAM capacities of 56K and 384 K bytes
- Three types of disk drives with capacities of 175K, 1.2M and 16M bytes
- Two dot matrix printers with 80 and 132 line capacity
- A Selectric typewriter interface and a daisy wheel printer

Match these to your exact need, add one or more of our intelligent terminals and put together a system from one source with guaranteed compatibility in both software and hardware.

Southwest Technical Products systems give you unmatched power, speed and versatility. They are packaged in custom designed woodgrain finished cabinets. Factory service and support on the entire system and local service is available in many cities.

SWTPC

SOUTHWEST TECHNICAL PRODUCTS CORPORATION
219 W. RHAPSODY
SAN ANTONIO, TEXAS 78216 (512) 344-0241

Technical Systems Consultants, Inc.

6800 & 8080 Text Processing

TSC's Text Editing System and Text Processing packages, available for both 6800 and 8080, can turn your micro into a very powerful word processing system. Two steps are involved. First you edit a file to say what you want without worrying about justification, pagination, etc. The TSC editor is a very powerful line and content oriented editor which may be used for this purpose.

The second step is to "process' the edited file. The TSC text processor reads the edited file and outputs it to a printer with right justification, pagination, page numbering and titling, indenting, margin control, title centering, and much more. Macros may be written in the text processor commands which allow complex operations such as footnotes, form letters, or almost anything imaginable.

Price includes manual with printed source listing and object code on tape or disk.

6800 Text Editing System

SL68-24C	**w/cassette**	**$30.45**
SL68-24D	**w/mini FLEX™ disk**	**$31.50**
SL68-24F	**w/8" FLEX™ 1.0 disk**	**$50.00**

8080 Text Editing System

SL80-10P	**w/paper tape**	**$37.50**
SL80-10F	**w/8" CP/M™ disk**	**$40.00**
SL80-10D	**w/5" HS CP/M™ disk**	**$40.00**

6800 Text Processing System

SL68-29C	**w/cassette**	**$38.95**
SL68-29D	**w/mini FLEX™ disk**	**$40.00**
SL68-29F	**w/8" FLEX™ 1.0 disk**	**$75.00**
SL68-29F2	**w/5" FLEX™ 2.0 disk**	**$65.00**

8080 Text Processing System

SL80-11P	**w/paper tape**	**$41.00**
SL80-11F	**w/8" CP/M™ disk**	**$50.00**
SL80-11D	**w/5" HS CP/M™ disk**	**$50.00**

FLEX™ Software

The FLEX™ disk operating system is rapidly becoming the standard of 6800 disk users. It's powerful capabilities and ease of use make it a natural choice for any application from the most minimal hobbyist system to the most complete industrial development system.

FLEX™ for SWTPC and SSB

SWTPc mini floppy users can now upgrade to FLEX™ 2.0, the same version of FLEX™ as found on the SWTPc 8" disk system. Simply add 8K of memory at $A000 and boot up from a FLEX™ 2.0 disk. Also available for Smoke Signal disk users in a mini floppy version (FLEX™ 2.0) and a full sized version (FLEX™ 1.0). Includes TSC editor and assembler on disk. All manuals also supplied (no source listings).

FX02-SWT	**FLEX™ 2.0 for 5" SWTPc**	**$75.00**
FX02-SSB	**FLEX™ 2.0 for 5" SSB**	**$90.00**
FX01-SSB	**FLEX™ 1.0 for 8" SSB**	**$100.00**

Sort/Merge Package

Sorts any type file according to parameters you supply. Files too large to fit in memory are automatically split up, individually sorted, and merged into one output file. Accepts fixed or variable length records, up to 20 ascending or descending keys, non-ASCII sequence sorts, and much more. No source listing. Specify mini FLEX™, FLEX™ 1.0, or FLEX™ 2.0

AP68-10	**6800 Sort/Merge**	**$75.00**
	Manual Only	**$15.00**

FLEX™ Utilities

Complete set of 36 additional utilities ranging from memory dump to disk diagnostics. Includes printed manuals and source listings with object code AND source listing on disk.

UV1-6D	**w/mini FLEX™ disk**	**$99.95**
UV1-6F	**w/8" FLEX™ 1.0 disk**	**$124.95**
UV1-6F2	**w/5" FLEX™ 2.0 disk**	**$119.95**

TSC BASIC for 6800

The fastest floating point BASIC interpreter for any 8 bit microprocessor. Features include six digit floating point math, full transcendental functions, unlimited string length, if/then/else construct, logical operators, and two-dimensional arrays including string arrays.

FLEX™ disk versions support random as well as sequential access files (the mini FLEX™ version has only sequential). Random files accessed by true record I/O or by virtual arrays. Disk version also allows TRACE, a renumber facility, ON ERROR GOTO, PTR for obtaining the address of a variable, CHAIN, and a very powerful method of printing to any type and number of output devices. SAVE and LOAD commands produce FLEX™ compatible files which can be externally edited while a COMPILE command places an unreadable, compressed form of the source on disk which can only be executed by RUN. This feature permits the distribution of proprietary BASIC programs.

Cassette version runs in a minimum of 12K, disk version should have at least 16K. No source listing is included.

AP68-11C	**w/cassette**	**$39.95**
AP68-11D	**w/mini FLEX™ disk**	**$49.95**
AP68-11F	**w/8" FLEX™ 1.0 disk**	**$59.95**
AP68-11F2	**w/5" FLEX™ 2.0 disk**	**$54.95**

All orders should include 3% for postage and handling (8% on foreign orders). Send 25¢ for a complete software catalog.

Technical Systems Consultants, Inc.

Box 2574 W. Lafayette, IN 47906

'68' MICRO JOURNAL

Portions of the text of '68' Micro Journal set using the following
6800/2, DMAF1 and CT-82
Southwest Technical Products Corp.
219 W. Rhapsody
San Antonio, TX 78216

Editor, Word Processor and Sort/Merge
Technical Systems Consultants, Inc.
Box 2574
W. LaFayette, IN 47906
*MINIFLEX & FLEX REG. ™ Technical Systems Consultants, Inc.

Selectric I/O
World Wide Electronics, Inc.
130 Northwestern Blvd.
Nashua, NH 03060

Publisher/Editor
Don Williams Sr.

Assistant Editor — Software
Mickey E. Ferguson

Assistant Editor — Hardware
Dennis Womack

Associate Editor — Southwest
Dr. Jack Bryant

Associate Editor — At Large
Dr. Chuck Adams

Associate Editor — Midwest
Howard Berenbon

Subscriptions and Office Manager
Joyce Williams

Typography and Color Separations
Williams Company, Inc.
Chattanooga, TN 37421

CONTENTS

Send All Correspondence To:

'68' Micro Journal
3018 Hamill Rd.
PO Box 849
Hixson, Tennessee 37343
— Phone —
Office: 615-870-1993
Plant: 615-892-7544

'68' Micro Journal is published 12 times a year by '68' Micro Journal, 6131 Airways Blvd., Chattanooga, TN 37421. Second Class postage paid at Chattanooga, TN. Postmaster: Send Form 3579 to '68' Micro Journal, PO Box 849, Hixson, TN 37343.

Increasing 40% September, '79 →

Subscription Rates U.S.A.:
(Special Charter Rate)
1 year $10.50
2 years $18.50
3 years $26.50
Lifetime $125.00 (one-time payment) Twice rate shown above for Air Mail/First Class anywhere in the U.S.A.

—ITEMS SUBMITTED FOR PUBLICATION—

(Letters to the Editor for Publication) All 'letters to the Editor' should be substantiated by facts. Opinions should be indicated as such. All letters must be signed. We are interested in receiving letters that will benefit or alert our readers. Praise as well as gripes is always good subject matter. Your name may be withheld upon request. If you have had a good experience with a 6800 vendor please put it in a letter. If the experience was bad put that in a letter also. Remember, if you tell us who they are then it is only fair that your name 'not' be withheld. This means that all letters published, of a critical nature, cannot have a name withheld. We will attempt to publish 'verbatim' letters that are composed using 'good taste.' We reserve the right to define (for '68' Micro) what constitutes 'good taste.'

(Articles and items submitted for publication) Please, always include your full name, address, and telephone number. Date and number all sheets. TYPE them if you can, poorly handwritten copy is sometimes the difference between go, no-go. All items should be on 8X11 inch, white paper. Most all art work will be reproduced photographically, this includes all listings, diagrams and other non-text material. All typewritten copy should be done with a NEW RIBBON. All hand drawn art should be black on white paper. Please no hand written code items over 50 bytes. Neatly typed copy will be directly reproduced. Column width should be 3¼ inches.

(Advertising) Any Classified: Maximum 20 words. All single letters and/or numbers will be considered one (1) word. No Commercial or Business Type Classified advertising. Classified ads will be published in our standard format. Classified ads $7.50 one time run, paid in advance.

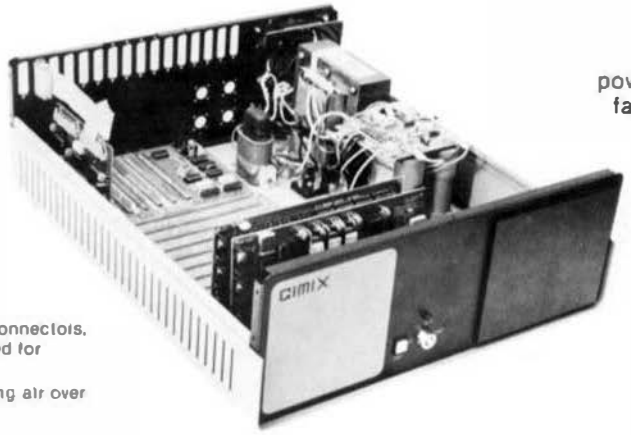
Commercial and/or Business advertisers please write or phone for current rate sheet and publication lag time.

WANT A CLASSY CHASSIS?

See GHOST Ad page 43

SYSTEM 68 by GIMIX

The Ultimate in SS 50 BUS Reliability



- A.C. fuse holder
- Removable A.C. cord
- Exhaust fan
- Punched for 16 D type date connectors, 4 video connectors and slotted for ribbon cables.
- Ventilation slots, direct cooling air over boards first.

Mainframe: includes chassis power supply switches, fan and mother board **$748.19**

16K Systems from $1294.29

Includes: Mainframe cabinet, mother board, power supply, fan, CPU, 16K static RAM, and choice of I/O card.

- Holds 2 5¼ disk drives (not included)
- GIMIX disk regulator cards (optional) mount on drives and wire to filter assembly board
- LED power indicator
- 3 position removable keyswitch (Off, Power On-Reset Off, Both On)
- Reset - can be locked out
- Optional filler plates (when no drives are used)

CABINET

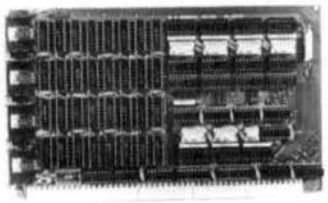
Heavyweight aluminum painted inside and out in grey and black baked enamel finish. Size: 18" wide x 21" deep x 7" high.

MOTHER BOARD

Hardware reconfigurable to give you the utmost versatility for use with various SS 50 bus software packages. *Gold plated* pins to insure long lasting electrical contact for protection against corrosion. Fifteen 50 pin slots plus eight DIP-switch addressable 30 pin I/O slots configurable to 4 or 8 decoded addresses. The fully buffered I/O block is addressable by DIP-switch to any 32 or 64 byte boundary and can also be disabled. UD1 and UD2 of the 50 pin bus can be strapped to UD3 and UD4 of the 30 pin bus. A fully shielded, (.090" thick), double sided P.C. board with noise reducing ground lines on the bottom side that separate all data, address, and signal lines, and a full ground plane on the top side. A 14 position clamping terminal block for all power and other external connections eliminates soldering, crimping or forming of wires.

POWER SUPPLY

Designed to power a fully loaded system plus two 5¼" disk drives -and keep running at constant voltage outputs even under adverse A.C. power input conditions. It consists of: *A 550 VA Ferro-resonant constant voltage transformer*, over 16 pounds of *brute force* custom designed for GIMIX to GIMIX' specs, an A.C. resonant capacitor, 3 D.C. filter capacitors, and GIMIX' unique filter assembly board that sits on top of the filter capacitors and includes individual fuses for each output, bleeder resistors, and a clamping terminal block for easy wiring connections. Almost a Quarter-farad of D.C. Filtering. *Brown-out and over-voltage insurance*. Supplies 8V at 25 Amps, + 15 Volts at 5 Amps and − 15 Volts at 5 Amps from A.C. input voltages ranging from 90 to 140 Volts.



16K Static RAM Boards for the SS-50 Bus $298.13

- Gold bus connectors
- 4 separate 4K Blocks
- Individual Addressing, Write Protect, and Enable/Disable for each Block

As above with Sockets and Software control features **368.16**

All GIMIX memory boards are assembled, Burnt-In for 2 weeks, and tested at 2 MHz

TI TMS 4044's — 10% SUPPLY

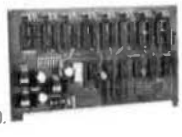
(Not an "equivalent", but the real thing!)

450 ns **$5.00 each** 250 ns **$6.00 each**

8K PROM BOARD $98.34

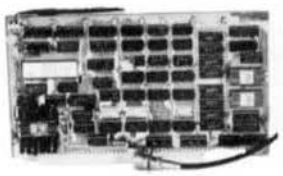
- Holds eight 2708 or 2708-compatible ROMS.
- DIP-switch addressable to any 8K boundary
- Gold Bus Connectors

2708s $7.90 each



Add $10. handling charge on orders under $200.

SS 50 BUS 80 X 24 VIDEO BOARD



Deluxe Version $458.76

Other Video Boards from $198.71

With hardware scrolling, x-y addressable cursor and multiple character generators. It includes a TMS 2716 EPROM that contains a full 128 upper and lower case ASCII character set with true descenders, plus a socket for another TMS 2716 for an optional 128 character set, plus 2K of RAM for user-defined programmable character sets. This gives the user the ability to create his own hieroglyphics, alphabet, graphic elements, etc., and store them on PROM, disk, or tape. The user can choose and intermix 384 different characters from any or all of the character generators and display up to 256 at one time, normally or inversely, and at full or half intensity, at any location on the screen. Contiguous 8x10 character cells permit solid lines and connecting patterns with user definable graphic elements.

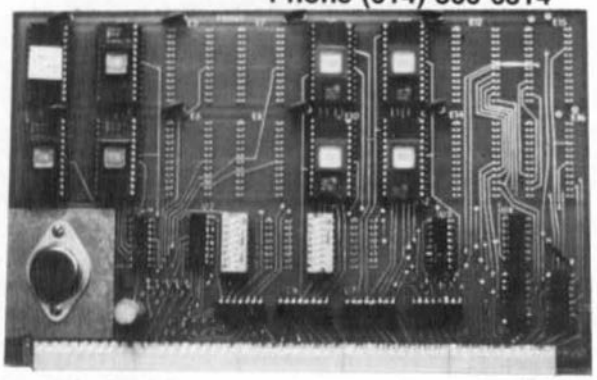
It is addressable to any 2K boundary. GHOSTable addressing allows multiple boards at the same address, making it ideal for multi-user applications. The available software includes a GMXBUG video based 3K ROM monitor, stand alone driver routines, and a program to create user defined characters.

Phone, write or see your dealer for complete brochure and price list.

GIMIX INC. 1337 WEST 37th PLACE
CHICAGO, ILLINOIS 60609
(312) 927-5510 • TWX 910-221-4055

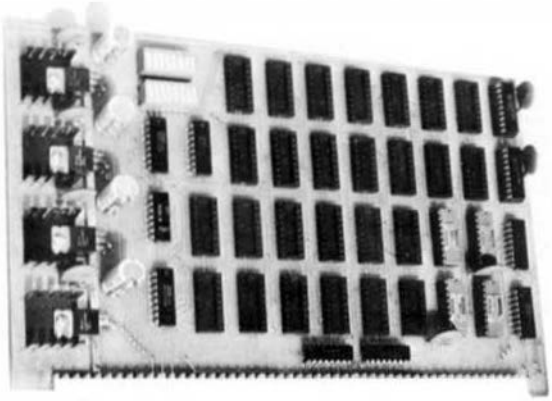
The Company that delivers.

Quality Electronic products since 1975.



DSD P/R-32K $27.00

32K or 16K EPROM & RAM memory card. 2716 2K x 8 or 2758 1K x 8 5V only EPROMS. TMS 4016 2K x 8 or MK 4118 1K x 8 5V RAMS. Up to 4 independent addressed 8K blocks. Dip switch or jumper selected. Size 9" x 5½"



DSD 2114-16K $27.00

Full Static 16K Ram memory card designed to use the 2114 or TMS 4045 1024 x 4 Static Ram. The card has two independent addressed 8K memory blocks. Card size 9" x 5½". Power requirements 7-8V unreg. @ 3.5A.

DSD U P 8255M $14.00

Universal parallel interface card with wire wrap area using INTEL'S 8255 parallel peripheral interface chip. 24 programmable I/O lines. (Three 8 bit Ports or Two 8 bit Ports with hand-shaking) Card size 5½" x 5" Standard SS-50 30 pin I/O BUS. 5V only.

Cards are bare with data and edge connector. Ohio residents add 4½% sales tax.

6847 Color Graphic card in design

OPTIMAL TECHNOLOGY, INC. A NEW PRODUCT ANNOUNCEMENT FOR EPROM PROGRAMMER SPANS 8 MICROCOMPUTERS AND PROGRAMS 1K, 2K, 4K EPROMS

Optimal Technology, Blue Wood 127, Earlysville, VA announces the EP-2A-79, EPROM Programmer. Software for programming and verifying programming is available for the 6800, 8080, Z-80, 8085, 6502 (KIM-1), F-8, 1802, and 2650 based computers. Packaged in a sloping panel aluminum case, the unit connects to microcomputer with a 14 pin ribbon cable thru 1 1/2 I/O ports. Software, supplied as a listing, requires approximately 256 bytes of RAM and includes instructions on how to relocate. Personality modules which plug into the front panel-mounted socket, are available for programming the 2708, 2716, TMS 2716, 2732, TMS 2708 and TMS 2532 EPROMS. Power requirements are 115 VAC 50/60 HZ at 15 watts. The EP-2A-79 is priced at $155.00 which includes 1 set of software. Personality Modules are priced at $15.00 except the Personality Modules for the 2732 and TMS 2532 which are $30.00. Available from stock. The EP-2A-79 6800 EPROM programmer uses 1 PIA (6820) with addresses 8004-8007 which is the same I/O as that available on the Motorola D2 Kit. The software is located on page 1, H'0100-01FF'. It is supplied as a commented source listing using the Motorola M6800 assembler format. Thus, the program is easy to relocate or change the PIA addresses. It is applicable to any 6800 computer system.

Additional information can be obtained from;

OPTIMAL TECHNOLOGY, INC.
BLUE WOOD 127
EARLYSVILLE, VA 22936
(804) 973-5482

CRUNCHERS CORNER

This monthly column is intended to provide a place for the exchange of ideas on microcomputer arithmetic. A systematic exposition of fixed and floating point arithmetic, hardware and software, algorithms for approximation and so on is planned. Questions and comments submitted to this column can be on any subject relevant to "number crunching," and should be addressed to:

Jack Bryant
Department of Mathematics
Texas A&M University
College Station, Texas 77843

We ask that all correspondents supply their names and addresses.

Earlier in this column programs were given to illustrate 16 bit two's complement arithmetic and conversions to and from ASCII sign-magnitude decimal representation. Because absolute addressing is simple, programs written this way are easiest to understand (and debug). By using page zero locations for software accumulators we also find the programs are fast. Last time, however, we noticed that this speed may be hard to use. That is because we have to move the numbers to and from accumulators, usually one byte at a time (although the index register may be available for two byte moves). For integers this may add as much as 24 bytes for each usage of one of these subroutines.

Last time we also noticed there are problems with the way we evaluate expressions. Although our program is only intended to test the arithmetic functions, it does contain logic to decode expressions. It is unconventional in several senses: first, like most assemblers, expressions are evaluated left-to-right with all operators having the same hierarchy or "rank". In an ordinary algebraic expression, / and * are performed before + and -: 3+4*2 is 9, not 14. Ordinary expressions get around the problem by the use of parentheses: (3+4)*2 *is* 14. The earlier program makes no provision for parentheses.

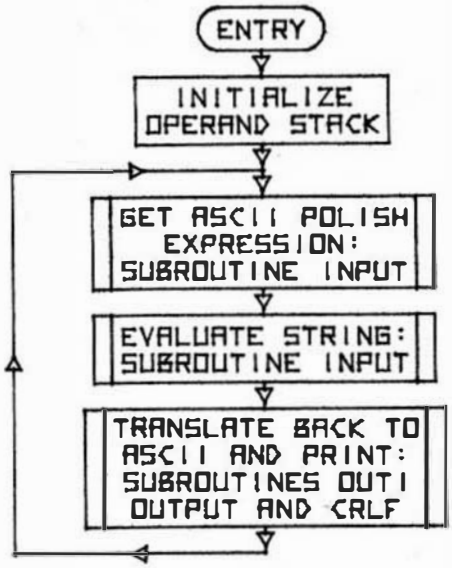
Expression evaluation is not per se a number crunching problem. It is really a logical problem. How this problem is handled, however, does affect number crunching. Wasteful operations in expression evaluation can easily defeat the most careful level programming. One way to avoid parenthesis as well as all that stuff about hierarchy was discussed last time: (reverse) Polish notation. Last month we saw some of the advantages to Polish over algebraic notation. In next month's column, we discuss programs to convert from algebraic to Polish notation, thus allowing the ideas from this month's program to be used in a high-level language such as BASIC. One of the many sources of BASIC's inefficiency is its repeated interpretive evaluation of algebraic expressions. A program to translate the expression (when the line is entered) into Polish (together with a lister program to translate Polish back to algebraic) would make this part of BASIC as efficient as it can be.

Before going on, observe that while 16 bit integers are inadequate for most real-world problems, they are adequate for addressing in a 6800 system. Thus the driver programs below could just as well be performing floating point operations. The "operand stack" would then contain addresses of

actual operands. Only the lowest level arithmetic functions would need to be changed. (Of course, a great many more operations would be introduced.)

THIS MONTH'S PROGRAM

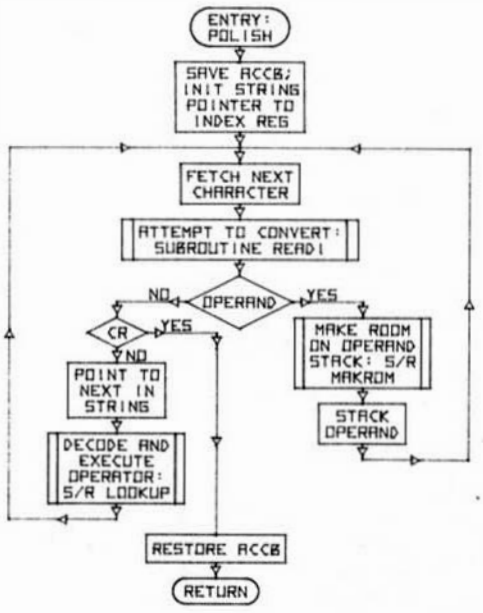
The main flow of control is seen in the following flow chart:



Subroutine INPUT is changed slightly from before to include back-space (ASCII $08) and cancel (ASCII $18). The main new program is the subroutine POLISH, which evaluates the ASCII string.

Subroutine LOOKUP was discussed in detail last month. (The current differs slightly; a *not found* error is flagged by ACCB = 0.) This month's subroutine READI differs because the unary minus sign has been replaced by 'C' (for 'Change Sign'). Also, if no operand is found, the index register is left equal to the contents of XTEMP and ACCA contains the operator. Finally, two subroutines are used to manage the stack: subroutine MAKROM insures that the operand stack does not eat the program. When the stack would overflow by stacking another operand, the entire stack is moved (with the bottom element being lost). This subroutine is also used by the E (for Enter) operator. Program DECCS2 is called before each binary operation to move the pointer up one operand, and to fix the stack, if necessary, so that this will always be possible.

The basic arithmetic subroutines have been rewritten to use indexed addressing exclusively. (That is, no absolute addressing is used.) They are all reentrant and, with one fixup, the entire package of subroutines is relocatable. (The fixup necessary is the error jump in the case of division by zero.) They are about the same length as the absolute addressing routines: MULI became 6 bytes shorter (17%), and DEVI-REMI is now 10 bytes shorter (5%). In this application a 30 character per second terminal is so much slower than the programs that even 47 divisions seem instantaneous.



The following table explains the symbols the program understands.

Symbol	Meaning	Use
0-9	Digits	Number entry.
C	Change Sign	
, or sp.	Delimiter	Seperates operands
BS	Backspace	During entry
CAN	Cancel	Start line over
X	Exchange	Interchange last two operands.
E	Enter	Duplicate last on stack.
D	Down	Move pointer down
A	Abs. value	Unary
N	Negative	operators
+	Addition	
-	Subtraction	Binary
*	Multiplication	operators
/	Division	
R	Remainder following division	
RET	Return	Translate, evaluate and print result.

SYMBOL TABLE:

ABSI	0356	ABSSTK	0354	ADDI	02F3	ADDSTK	02F1
ATEMP	00F7	BUFEND	00FE	COMMA	047D	CRLF	025C
CRLFD	04A1	CURSTK	00FA	CVT16	00F4	DECCOK	0346
DECCS2	0330	DIVI	036A	DIVINE	03BC	DIVSTK	0368
DLOOP	0401	DLOOPM	0415	DNEG	0431	DNOERR	0390
DOVF	03C8	DPOS1	039D	DPOS3	03A6	DPOS5	03B4
DPOS6	03BA	DSHFT	0426	ENDSTK	00FC	ENTER	047E
EXGSTK	037F	FIXSGN	03AE	FOUND	029D	IBCKS	0246
ICANC	0224	IFULL	024C	INBUF	00EF	INEEE	E1AC
INEXT	0227	INPUT	0221	ISTASH	0236	LOOKUP	0291
MAGN	0318	MAKROM	049B	MLOOP	0495	MSKIP	0322
MSTASH	049E	MULI	0311	MULSTK	030F	NEGFLG	00F3
NEGI	035D	NEGI1	0365	NEGIEN	035F	NEGM	0349
NEGM1	0351	NEGMEN	034B	NEGSTK	035B	NEXT	02A8
NOTONE	02D4	OLOOP	0442	OOK	0450	OPLUS	046E
OPRTAB	04A4	OQUIT	0266	OUTEEE	E1D1	OUTI	0435
OUTPUT	0251	PDATA1	E07E	PSON	0287	PLOOP2	026A
PNONUM	0281	POLISH	0267	RCHGS	02CF	RDIV	03F9
READI	02A1	REMI	0387	REMSTK	0385	RFOUND	00F6
RINX	02CC	RNONU1	02EC	RNONUM	02EE	SKIP	02E8
SUBI	0302	SUBSTK	0300	SWAPIM	036C	TABEND	02ED
TEST	0210	TOPSTK	00F8	USRERR	04C9	XTEMP	00F1

```
INTEGER ARITHMETIC PACKAGE V 3.0        '68' MICRO JOURNAL        PAGE 1

                        *       MIKBUG EQUATES
                        *
E07E                    PDATA1  EQU     $E07E   BLOCK OUTPUT
E1AC                    INEEE   EQU     $E1AC   INPUT ONE CHARACTER
E1D1                    OUTEEE  EQU     $E1D1   CHARACTER OUTPUT ROUTINE
                        *
                        *       PAGE ZERO CONSTANTS
                        *
00FF                    ORG     $FF
00EF 01 00      INBUF   FDB     $100    INPUT BUFFER BEGINNING
00F1            XTEMP   RMB     2       TEMPORARY STORAGE FOR INDEX REGISTER
00F3            NEGFLG  RMB     1       USED IN CONVERSION. ASCII TO BINARY.
00F4            CVT16   RMB     2       DITTO
00F6            PFOUND  RMB     1       DITTO
00F7            ATEMP   RMB     1       DITTO  NOTE: CONVERSIONS MUST BE FAST!
00F8 01 01      TOPSTK  FDB     $0101   TOP OF STACK.
00FA            CURSTK  RMB     2       CURRENT TWO OPERAND POINTER.
00FC 01 E1      ENDSTK  FDB     $01E1   ALLOW 48 OPERANDS.
00FE 01 FF      BUFEND  FDB     $01FF   END OF BUFFER AREA
                        *
                        *       INDEXED REGISTER ADDRESSED STORAGE ON PAGE 1
                        *
0200                    ORG     $0200   TEST ROUTINE STARTS HERE
                        *
                        *       TEST PROGRAM FOR INTEGER ARITHMETIC PACKAGE
                        *               VERSION 3.0
                        *
0200 BD E0 5A           START   JSR     CRLF    START WITH A NEW LINE.
0203 DE F8              LDX     TOPSTK  INITIALIZE OPERAND STACK.
0205 6F 00              CLR     0,X
0207 6F 01              CLR     1,X     CLEAR SPACE FOR TWO OPERANDS
0209 6F 02              CLR     2,X
020B 6F 03              CLR     3,X
020D 08                 INX             PREPARE TO SET OP STACK POINTER
020E 08                 INX
020F DF F8              STX     CURSTK
0211 BD 0F      TEST    JSR     INPUT   GET NUMBER AND OPERATOR STRING
0214 DE EF              LDX     INBUF
0216 BD 51              JSR     POLISH  PERFORM OPERATIONS
0218 DE FE              LDX     BUFEND
021A BD 04 35           JSR     OUTI
021D BD 30              JSR     OUTPUT  PRINT THE RESULT.
021F BD 3D              JSR     CRLF    AND START A NEW LINE
0221 20 EF              BRA     TEST    DO IT AGAIN
                        *
                        *       INPUT SUBROUTINE
                        *
                        *               SUBROUTINE TO TAKE A STRING FROM THE CONSOLE
                        *               AND STORE IN INPUT BUFFER.

INTEGER ARITHMETIC PACKAGE V 3.0        '68' MICRO JOURNAL        PAGE 2

                        *
0223 DF F1      INPUT   STX     XTEMP   SAVE INDEX REGISTER
0225 37                 PSH B           SAVE ACCB
0226 5F                 CLR B           SET COUNTER
0227 DE EF      ICANC   LDX     INBUF   INPUT BUFFER STARTS AT <INBUF>
0229 BD E1 AC   INEXT   JSR     INEEE   GET A CHARACTER
022C 81 08              CMP A #$08      BACKSPACE?
022E 27 10              BEQ     IBCKS
0230 81 18              CMP A #$18      CANCEL?
0232 26 04              BNE     ISTASH
0234 BD 3D              BSR     CRLF    NEW LINE
0236 20 EE              BRA     ICANC
0238 A7 00      ISTASH  STA A 0,X       STORE IT
023A 08                 INX             BUMP COUNTER
023B 5C                 INC B
023C 29 10              BVS     IFULL   BUFFER SIZE = 127
023E 81 0D              CMP A #$D       CR?
0240 26 E7              BNE     INEXT   NOPE
0242 BD 1A              BSR     CRLF    NEW LINE.
0244 DE F1              LDX     XTEMP   RESTORE INDEX REGISTER
0246 39                 PUL B           RESTORE ACCB
0247 39                 RTS
0248 5A         IBCKS   DEC B           UNCOUNT
0249 2F DA              BLE     ICANC   NIL IN BUFFER
024B 09                 DEX             UNPOINT
024C 20 99              BRA     INEXT
024E 09         IFULL   DEX             BUFFER FULL: GENERATE CR.
024F 86 0D              LDA A #$D
0251 20 E5              BRA     ISTASH
                        *
                        *       OUTPUT SUBROUTINE
                        *
                        *               THIS SUBROUTINE PRINTS A STRING
                        *               POINTED TO BY THE INDEX REGISTER
                        *               WITH ACCB CONTAINING THE TOTAL
                        *               COUNT OF THE NUMBER OF CHARACTERS
                        *               PRINTED.  ACCB IS ZERO ON EXIT.
                        *               AND THE INDEX REGISTER IS LEFT
                        *               POINTING TO THE LAST PRINTED PLUS 1.
                        *
0253 5A         OUTPUT  DEC B           BEGIN WITH A COUNT
0254 2D 12              BLT     OOUIT
0256 A6 00              LDA A 0,X
0258 BD E1 D1           JSR     OUTEEE  PRINT IT
025B 08                 INX             POINT TO THE NEXT ONE
025C 20 F5              BRA     OUTPUT
                        *
                        *       SUBROUTINE TO PRINT A CR LF.
                        *
025E DF F1      CRLF    STX     XTEMP

INTEGER ARITHMETIC PACKAGE V 3.0        '68' MICRO JOURNAL        PAGE 3

0260 CE 04 A1           LDX     #CRLFD
0263 BD E0 7E           JSR     PDATA1
0266 DE F1              LDX     XTEMP
0268 39         OOUIT   RTS
                        *
                        *       SUBROUTINE POLISH
                        *
                        *               THIS SUBROUTINE EVALUATES A STRING POINTED TO BY
                        *               THE INDEX REGISTER AS A REVERSE POLISH
                        *               EXPRESSION.  SEE TEXT FOR DETAILS.
                        *
                        *       ON ENTRY, INDEX REGISTER POINTS TO THE START
                        *               OF THE STRING.
                        *
                        *       EXIT BY NORMAL RTS IS TAKEN WHEN THE FIRST CR
                        *               IS FOUND.
                        *
0269 37         POLISH  PSH B
026A DF F1              STX     XTEMP   NOTE: NONE OF THE S/R CALLED CAN USE XTEMP.
```

```
INTEGER ARITHMETIC PACKAGE V 3.0        '68' MICRO JOURNAL        PAGE 4

026C DE F1      PLOOP2  LDX     XTEMP
026E BD 33              BSR     READI   TRY TO CONVERT A NUMBER
0270 2C 0F              BGE     PNONUM
0272 DF F1              STX     XTEMP   NUMBER FOUND
0274 BD 04 6B           JSR     MAKROM  MAKE ROOM ON THE STACK
0277 96 F4              LDA A CVT16     ROOM (NOW)... STORE THE NUMBER
0279 A7 00              STA A 0,X
027B 96 F5              LDA A CVT16+1
027D A7 01              STA A 1,X
027F 20 E9              BRA     PLOOP2
                        *
0281 81 0D      PNONUM  CMP A #$D       NOTE: ACCA CONTAINS THE OPERATOR
0283 26 02              BNE     PGON
0285 33                 PUL B
0286 39                 RTS
0287 08         PGON    INX
0288 DF F1              STX     XTEMP   UPDATE POINTER
028A CE 04 A4           LDX     #OPRTAB POINT TO OPERATOR TABLE
028D BD 02              JSR     LOOKUP
028F 20 DA              BRA     PLOOP2  NOTE THAT ERRORS ARE IGNORED
                        *
                        *       SUBROUTINE LOOKUP
                        *
                        *               SUBROUTINE TO DO A TABLE LOOKUP FOR A MATCH
                        *               BETWEEN THE CONTENTS OF ACCA AND THE TABLE
                        *               POINTED TO BY THE INDEX REGISTER.  ACCB IS
                        *               LEFT = 0 IF NO MATCH IS FOUND; OTHERWISE,
                        *               ACCB = ACCA ON EXIT.
                        *
                        *               NORMAL EXIT IS AN INDEXED JUMP TO THE MATCHING

INTEGER ARITHMETIC PACKAGE V 3.0        '68' MICRO JOURNAL        PAGE 4

                        *               SUBROUTINE (WHICH THUS MUST TERMINATE IN  RTS ).
                        *
                        *       NOTE:   IT IS ASSUMED THAT EACH  S/R  DOES NOT CHANGE ACCB.
                        *
0291 E6 00      LOOKUP  LDA B 0,X
0293 27 58              BEQ     TABEND
0295 11                 CBA
0296 27 05              BEQ     FOUND
0298 08                 INX
0299 08                 INX
029A 08                 INX
029B 20 F4              BRA     LOOKUP
029D EE 01      FOUND   LDX     1,X
029F 6E 00              JMP     0,X
                        *
                        *       READ SUBROUTINE
                        *
                        *               ENTER WITH INDEX REGISTER POINTING TO STRING
                        *               TO BE CONVERTED TO 16 BIT BINARY.
                        *
                        *               EXIT WITH INDEX REGISTER POINTING TO NEXT NON
                        *               (0-9) OR CHANGE SIGN (ASCII '-') AND WITH NUMBER
                        *               IN  CVT16
                        *
                        *       NOTE:   THE INDEX REGISTER IS NOT CHANGED IF NO NUMBER
                        *               IS FOUND.
                        *
                        *       NOTE:   EACH VALID NUMBER MUST CONTAIN AN ACTUAL DECIMAL
                        *               DIGIT I.E. CHS + '-' WILL NOT WORK ALONE.
                        *
                        *       NOTE:   THE VALUE OF ACCA IS THE NON 0-9 OR -
                        *               FOUND IN CASE NO NUMBER IS THERE.
                        *
02A1 37         READI   PSH B           SAVE ACCB
02A2 5F                 CLR B           CLEAR ACCUMULATOR
02A3 D7 F6              STA B PFOUND
02A5 37                 PSH B
02A6 D7 F3              STA B NEGFLG    CLEAR NEGATIVE FLAG
02A8 A6 00      NEXT    LDA A 0,X       DO IT
02AA 81 30              CMP A #$30      '0' ZERO?
02AC 2D 26              BLT     NOTONE
02AE 81 39              CMP A #$39      '9' NINE?
02B0 2E 22              BGT     NOTONE
02B2 97 F6              STA A PFOUND
02B4 80 30              SUB A #$30      GOT ONE, CONVERT TO BINARY
02B6 97 F7              STA A ATEMP     SAVE FOR LATER
02B8 32                 PUL A           RESTORE MSB
02B9 38                 ASL B           *
02BA 49                 ROL A           *
```

```
INTEGER ARITHMETIC PACKAGE V 3.0        '68' MICRO JOURNAL        PAGE 5

02BB 97 F4              STA A CVT16     *
02BD D7 F5              STA B CVT16+1   *
02BF 38                 ASL B           *   MULTIPLY BY 10
02C0 49                 ROL A           *
02C1 58                 ASL B           *
02C2 49                 ROL A           *
02C3 DB F5              ADD B CVT16+1   *
02C5 99 F4              ADC A CVT16     *
02C7 DB F7              ADD B ATEMP     ADD IN DIGIT
02C9 89 00              ADC A #0        AND CARRY OUT.
02CB 36                 PSH A           FREE ACCA
02CC 09         RINX    INX
02CD 20 D9              BRA     NEXT    DO IT AGAIN
02CF 73 00 F3   RCHGS   COM     NEGFLG
02D2 20 F8              BRA     RINX
02D4 81 43      NOTONE  CMP A #'C       '-'
02D6 27 F7              BEQ     RCHGS   YES A CHS OPERATOR
02D8 7D 00 F6           TST     PFOUND  SEE IF ANY DIGIT SINCE START
02DB 27 11              BEQ     PNONUM
02DD 32                 PUL A           NEED AT LEAST ONE DIGIT WAS FOUND
02DE 7D 00 F3           TST     NEGFLG  NEGATIVE?
02E1 27 05              BEQ     SKIP
02E3 43                 COM A
02E4 50                 NEG B           MAKE IT NEGATIVE
02E5 26 01              BNE     SKIP
02E7 4C                 INC A           CARRY OUT
02E8 97 F4      SKIP    STA A CVT16     STASH THE NUMBER
02EA D7 F5              STA B CVT16+1   WE BUILT.
02EC 33         PNONUI  PUL B           RESTORE ACCB
02ED 39         TABEND  RTS             FUNCTIONS AS RETURN FOR LOOKUP
02EE 33         PNONUM  PUL B           FIX UP STACK
02EF 20 FB              BRA     PNONUI
                        *
02F1 BD 3D      ADDSTK  BSR     DECCS2
                        *
                        *       SUBROUTINE ADDI
                        *
                        *               ADDS 2,X TO 0,X AND PLACES THE RESULT IN 0,X.
                        *
```

```
02F3 A6 01      ADD1    LDA A 1,X
02F5 AB 02              ADD A 3,X
02F7 A7 01              STA A 1,X
02F9 A6 00              LDA A 0,X
02FB A9 02              ADC A 2,X
02FD A7 00              STA A 0,X
02FF 39                 RTS
                *
0300 8D 2E      SUBSTK  BSR   DECCS2
                *
                *       SUBROUTINE SUB1
```



```
                *
                *       SUBTRACTS 2,X FROM 0,X. LEAVING THE RESULT IN 0,X.
                *
0302 A6 01      SUB1    LDA A 1,X
0304 A0 03              SUB A 3,X
0306 A7 01              STA A 1,X
0308 A6 00              LDA A 0,X
030A A2 02              SBC A 2,X
030C A7 00              STA A 0,X
030E 39                 RTS
                *
030F 8D 1F      MULSTK  BSR   DECCS2
                *
                *       SUBROUTINE MUL1
                *
                *       MULTIPLIES 0,X BY 2,X AND PLACES THE RESULT 0,X.
                *
0311 37         MUL1    PSH B           SAVE ACCB
0312 C6 10              LDA B #16       SET COUNTER
0314 E7 04              STA B 4,X
0316 4F                 CLR A           CLEAR PRODUCT MSB
0317 5F                 CLR B           AND LSB
0318 66 02      MAGN    ROR   2,X
031A 66 03              POR   3,X       TEST LEAST SIGNIFICANT BIT
031C 24 04              BCC   MSHIP     OF MULTIPLIER.
031E EB 01              ADD B 1,X       ADD IN MULTIPLICAND
0320 A9 00              ADC A 0,X         IF SET
0322 68 01      MSHIP   ASL   1,X       SHIFT MULTIPLICAND
0324 69 00              ROL   0,X       FOR NEXT TIME
0326 6A 04              DEC   4,X       COUNT ONE
0328 26 EE              BNE   MAGN      REPEAT IF NOT FINISHED
032A E7 01              STA B 1,X       STORE PRODUCT
032C A7 00              STA A 0,X
032E 33                 PUL B
032F 39                 RTS
                *
                *       SUBROUTINE TO PREPARE FOR A BINARY OPERATION
                *
0330 BE FA      DECCS2  LDX   CURSTK
0332 09                 DEX
0333 09                 DEX
0334 9C F8              CPX   TOPSTK
0336 26 0E              BNE   DECCRA
0338 BF FA              LDX   CURSTK    GENERATE ZERO ON TOP OF STACK
033A 4A 00              LDA A 0,X
033C A7 02              STA A 2,X       AND MOVE STACK DOWN
033E A6 01              LDA A 1,X
0340 A7 03              STA A 3,X
0342 6F 00              CLR   0,X
0344 6F 01              CLR   1,X
```



```
0346 BF FA      DECCRA  STX   CURSTK
0348 39                 RTS
                *
                *       SUBROUTINE NEGM
                *
                *       REPLACES 2,X BY -2,X
                *
0349 A6 02      NEGM    LDA A 2,X
034B 43         NEGMEN  COM A           ENTRY FOR WHEN 2,X ALREADY LOADED
034C 60 03              NEG   3,X
034E 26 01              BNE   NEGMI
0350 4C                 INC A
0351 A7 02      NPGMI   STA A 2,X
0353 39                 RTS
                *
0354 BE FA      ABSSTK  LDX   CURSTK
                *
                *       SUBROUTINE IABS
                *
                *       REPLACES 0,X BY ABSOLUTE VALUE OF 0,X.
                *
0356 A6 00      ABS1    LDA A 0,X
0358 2B 05              BLT   NEGIEN    IF NEGATIVE MAKE IT POSITIVE.
035A 39                 RTS
                *
035B BE FA      NEGSTK  LDX   CURSTK
                *
                *       SUBROUTINE NEG1
                *
                *       REPLACES 0,X BY -0,X
                *
035D A6 00      NEG1    LDA A 0,X
035F 43         NEGIEN  COM A           ENTRY FOR WHEN A ALREADY LOADED.
0360 60 01              NEG   1,X
0362 26 01              BNE   NEGI1
0364 4C                 INC A
0365 A7 00      NEGI1   STA A 0,X
0367 39                 RTS
                *
0368 8D C6      DIVSTK  BSR   DECCS2
                *
                *       SUBROUTINE DIV1
                *
                *          REPLACES 0,X BY 0,X/2,X.  REMAINDER IS IN 2,X.
                *
036A 8D 1B      DIV1    BSR   REM1      PERFORM DIVISION
                *
                *       SUBROUTINE TO INTERCHANGE LAST TWO OPERANDS ON STACK
                *
                *          CALL BY
```



```
                *       LDX   CURSTK
                *       DEX
                *       DEX
                *       BSR   SWAPIN
                *       LDX   CURSTK  (IF NECESSARY)
                *
036C 37         SWAPIN  PSH B
036D A6 00              LDA A 0,X
036F E6 02              LDA B 2,X
0371 E7 00              STA B 0,X
0373 A7 02              STA A 2,X
0375 A6 01              LDA A 1,X
0377 E6 03              LDA B 3,X
0379 A7 03              STA A 3,X
037B E7 01              STA B 1,X
037D 33                 PUL B
037E 39                 RTS
                *
037F DE FA      EMGSTK  LDX   CURSTK
0381 09                 DEX
0382 09                 DEX
0383 20 E7              BRA   SWAPIN
                *
0385 8D A9      REMSTK  BSR   DECCS2
                *
                *       SUBROUTINE REM1
                *
                *
                *          REPLACES 0,X BY THE REMAINDER FOLLOWING A DIVISION
                *       0,X/2,X. 2,X CONTAINS THE QUOTIENT.
                *
                *       INDEXED STORAGE:
                *
                *       ENTRY           EXIT
                *
                * 0,1   DIVND     REMAINDER
                * 2,3   DIVSR     QUOTIENT
                * 4,5   SHIFT AREA
                * 6,7   TEMP STORAGE
                * 8,10  NEGATIVE FLAGS
                *  9    COUNTER
                *
0387 A6 03      REM1    LDA A 3,X
0389 AA 02              ORA A 2,X       CHECK DIVISION BY ZERO
038B 26 03              BNE   DNDERR
038D 7E 04 C9           JMP   VIOERR    INT DIV BY ZERO CODE = 9
0390 37         INDERR  PSH B           SAVE ACCB
0391 5F                 CLR B           CLEAR NEGATIVE QUOTIENT FLAG.
0392 E7 08              STA B 8,X       CLEAR NEGATIVE REMAINDER FLAG.
0394 A6 00              LDA A 0,X         CHECK NUMERATOR SIGN.
0396 2C 05              BGE   DPOS1
```



```
0398 53                 COM B           SET NOF
0399 E7 08              STA B 8,X       SET NOF
039B EB E2              ADD B NEGIEN    MAKE DIVIDEND POSITIVE
039D A6 02      DPOS1   LDA A 2,X       CHECK DIVISOR.
039F 2C 05              BGE   DPOS2
03A1 53                 COM B           TOGGLE NOF
03A2 8D A7              BSR   NEGMEN    MAKE DIVISOR POSITIVE.
03A4 2D 16              BLT   DIVIDE    DON'T DIVIDE IF OVERFLOW.
03A6 E7 0A      DPOS2   STA B 10,X      NOW DIVIDE!
03A8 E6 09              LDA B 9,X       CHECK DIVIDEND OVERFLOW.
03AA 2B 1C              BMI   DOVF
03AC 8D 4A              BSR   PDIV      OK TO DIVIDE.
                *                       FIX SIGNS NOW
03AE E6 0A      FIXSGN  LDA B 10,X
03B0 27 02              BEQ   DPOS3
03B2 8D 95              BSR   NEG1      MAKE QUO POS.
03B4 E6 08      DPOS3   LDA B 8,X
03B6 27 02              BEQ   DPOS4
03B8 8D A3              BSR   NEG1      MAKE REMAINDER NEGATIVE.
03BA 33         DPOS4   PUL B
03BB 39                 RTS
                *
03BC 6F 02      DIVIDE  CLR   2,X       NOTE 3,X = 0
03BE E6 00              LDA B 0,X
03C0 2C F2              BGE   DPOS3     OK. GO FIX REMAINDER SIGN.
03C2 6C 03              INC   3,X       TWO OVERFLOWS.
03C4 6F 00              CLR   0,X       ZERO REMAINDER.
03C6 20 F2              BRA   DPOS4     1 QUOTIENT.
03C8 86 40      DOVF    LDA A #$40      BINARY 0100 0000
03CA A7 00              STA A 0,X       MAKE 0,X = -(-32768.2)
03CC A6 02              LDA A 2,X
03CE A7 06              STA A 6,X       SAVE DIVISOR FOR LATER.
03D0 A6 03              LDA A 3,X
03D2 A7 07              STA A 7,X
03D4 8D 23              BSR   PDIV      NOW DIVIDE.
03D6 68 03              ASL   3,X       MULTIPLY QUO BY 2
03D8 69 02              ROL   2,X
03DA 69 01              ASL   1,X       MULTIPLY REM BY 2
03DC 69 00              ROL   0,X
03DE A6 00              LDA A 0,X
03E0 E6 01              LDA B 1,X
03E2 E0 07              SUB B 7,X
03E4 A2 06              SBC A 6,X       SEE IF DIVISION WAS CORRECT.
03E6 2B C6              BLT   FIXSGN
03E8 E7 01              STA B 1,X       NO. FIX REMAINDER.
03EA A7 00              STA A 0,X
03EC A6 02              LDA A 2,X       AND FIX QUOTIENT.
03EE E6 03              LDA B 3,X
03F0 5C                 INC B
03F1 89 00              ADC A #0
```



```
03F3 E7 03              STA B 3,X
03F5 A7 02              STA A 2,X
03F7 20 B5              BRA   FIXSGN    GO CORRECT SIGNS.
                *
                *
                *       MAIN LOOP(S) TO PERFORM ACTUAL DIVISION
                *       WHEN EVERYTHING IS IN ORDER.
                *
03F9 C6 10      PDIV    LDA B #16
03FB E7 09              STA B 9,X       SET LOOP COUNTER.
03FD 6F 04              CLR   4,X       CLEAR DIVISOR SHIFT AREA.
03FF 6F 05              CLR   5,X
0401 64 02      DLOOP   LSR   2,X
0403 66 03              ROR   3,X       SHIFT DIVISOR RIGHT.
0405 66 04              ROR   4,X
0407 66 05              ROR   5,X
0409 6A 09              DEC   9,X       COUNT ONE.
040B A6 02              LDA A 2,X         CHECK IF UPPER 16 BITS ZERO.
040D AA 03              ORA A 3,X
040F 26 F0              BNE   DLOOP
0411 64 04              LSR   4,X       BEGIN WITH A SHIFT
0413 66 05              ROR   5,X
0415 A6 01      DLOOPM  LDA A 1,X       MAIN LOOP
0417 A0 05              SUB A 5,X
0419 E6 00              LDA B 0,X
041B E2 04              SBC B 4,X
041D 2D 12              BLT   DNEG      TEST NEGATIVE
041F E7 00              STA B 0,X       NO. POSITIVE.
0421 A7 01              STA A 1,X       FIX DIVIDEND
```

```
0423 0D                 SEC
0424 69 03              ROL   3,X       AND DUPLICATE.
0426 69 02      DIHFT   ROL   2,X
0428 67 04              ASR   4,X       SHIFT DIVISOR
042A 66 05              ROR   5,X
042C 6A 09              DEC   9,X       COUNT
042E 26 E5              BNE   DLOOPA
0430 39                 RTS
0431 68 03      DNEG    NEG   3,X
0433 20 F1              BRA   DIHFT

                *
                *       SUBROUTINE OUTI
                *
                *             SUBROUTINE TO CONVERT FROM 16 BIT
                *             TWO'S COMPLEMENT BINARY TO ASCII
                *             SIGN-MAGNITUDE DECIMAL REPRESENTATION.
                *             THE NUMBER IS POINTED TO BY <VPSTK>.
                *
                *             ON ENTRY:  INDEX REGISTER POINTS
                *                     TO END OF AT LEAST 6 BYTE BUFFER
                *                     AREA TO RECIEVE THE ASCII STRING.
                *
                *             ON EXIT:  INDEX REGISTER

INTEGER ARITHMETIC PACKAGE V 3.0        '68' MICRO JOURNAL        PAGE 11

                *                     POINTS TO THE START OF THE STRING.
                *             ACCB CONTAINS THE COUNT OF THE NUMBER
                *                     OF BYTES IN THE OUTPUT STRING.
                *
0435 5F         OUTI    CLR B           COUNTER
0436 DF F1              STX   XTEMP
0438 DE FA              LDX   CURSTK
043A A6 00              LDA A 0,X
043C 97 F9              STA A ATEMP     SAVE SIGN
043E A6 01              LDA A 1,X
0440 97 F4              STA A CVT16     SAVE NUMBER
0442 86 0A      DLOOP   LDA A #10
0444 A7 03              STA A 3,X
0446 4F 02              CLR   2,X       LOAD DIVISOR WITH 10
0448 BD 03 6A           JSR   DIVI      DIVIDE
044B A6 03              LDA A 3,X       REMAINDER IN 3,X
044D 2C 01              BGE   OOK
044F 40                 NEG A
0450 8B 30      OOK     ADD A #$30      MAKE ASCII
0452 DE F1              LDX   XTEMP
0454 A7 00              STA A 0,X
0456 5C                 INC B           COUNT ONE
0457 09                 DEX             POINT TO NEXT SLOT
0458 DF F1              STX   XTEMP     FREE INDEX REGISTER
045A DE FA              LDX   CURSTK
045C A6 00              LDA A 0,X
045E 4A 01              DEC A 1,X       IS IT ZERO YET?
0460 26 E0              BNE   DLOOP
0462 DE F1              LDX   XTEMP
0464 9A F9              LDA A ATEMP     WRAP UP
0466 2C 06              BGE   DPLUS
0468 86 2D              LDA A #$2D      MINUS SIGN
046A A7 00              STA A 0,X
046C 09                 DEX
046D 5C                 INC B
046E 08         DPLUS   INX             POINT TO FIRST ONE
046F DF F1              STX   XTEMP
0471 DE FA              LDX   CURSTK
0473 96 F7              LDA A ATEMP
0475 A7 00              STA A 0,X       RESTORE THE NUMBER ON STACK
0477 96 F4              LDA A CVT16
0479 A7 01              STA A 1,X
047B DE F1              LDX   XTEMP
047D 39         COMMA   RTS             COMMA: SEPERATE OPERANDS
                *
                *       SUBROUTINE TO DUPLICATE LAST ENTRY ON STACK IF POSSIBLE
                *
047E 8D 0A      ENTER   BSR   MAKROM    MAKE ROOM FOR ANOTHER MEMBER OF THE STACK
0480 09                 DEX
0481 09                 DEX             POINT TO THE OLD ONE

INTEGER ARITHMETIC PACKAGE V 3.0        '68' MICRO JOURNAL        PAGE 12

0482 A6 00              LDA A 0,X
0484 A7 02              STA A 2,X       AND MOVE IT DOWN
0486 A6 01              LDA A 1,X
0488 A7 03              STA A 3,X
048A 39                 RTS
                *
                *       SUBROUTINE TO EXTEND THE STACK BY ONE OPERAND
                *       IF THERE IS ROOM ON THE STACK.
                *       OTHERWISE, MOVE THE STACK TOWARD TOPSTK.  ON EXIT,
                *       CURSTK POINTS TO A VALID POSITION ON THE STACK.
                *
                *       ALSO: THIS VALUE IS PRESENT IN THE INDEX REGISTER.
                *
048B 9E FA      MAKROM  LDX   CURSTK
048D 08                 INX
048E 08                 INX
048F 9C FC              CPX   ENDSTK    SEE IF WE'RE OUT OF ROOM
0491 26 05              BNE   MSTACK
0493 9E F9              LDX   TOPSTK    YES. MOVE THE STACK
0495 A6 02      MLOOP   LDA A 2,X
0497 A7 00              STA A 0,X
0499 08                 INX
049A 9C FA              CPX   CURSTK
049C 26 F7              BNE   MLOOP
049E DF FA      MSTACK  STX   CURSTK    ALL MOVED
04A0 39                 RTS
                *
                *
                *                       END OF CODE.
04A1 0D         CRLFB   FCB   $D,$A,4   DATA FOR CRLF
04A2 0A 04
                *
                *       OPERATOR JUMP TABLE
                *
04A4 2B         OPERTAB FCB   '+
04A5 A2 F1              FDB   ADD
04A7 2D                 FCB   '-
04A8 A3 00              FDB   SUBSTK
04AA 41                 FCB   'A
04AB A3 54              FDB   ABSSTK
04AD 4E                 FCB   'N
04AE 03 8B              FDB   NEGSTK
04B0 2A                 FCB   '*
04B1 04 7D              FDB   COMMA
```

```
04B3 2C                 FCB   $2C
04B4 04 7D              FDB   COMMA
04B6 45                 FCB   'E

INTEGER ARITHMETIC PACKAGE V 3.0        '68' MICRO JOURNAL        PAGE 13

04B7 04 7E              FDB   ENTER
04B9 2A                 FCB   $2A
04BA 02 0F              FDB   MULTIP
04BC 2F                 FCB   $2F
04BD 03 6B              FDB   DIVSTK
04BF 52                 FCB   'R
04C0 03 85              FDB   REMSTK
04C2 44                 FCB   'D
04C3 03 30              FDB   DFCCT2    MOVES POINTER DOWN TO NEXT OPERAND
04C5 58                 FCB   'X
04C6 03 2F              FDB   EXGSTK
04C8 00                 FCB   0         END OF TABLE
                *
04C9 39         USERRP  RTS             USER SUPPLIED ERROR ROUTINE.

INTEGER ARITHMETIC PACKAGE V 3.0        '68' MICRO JOURNAL        PAGE 14

                        END
      NO ERROR(S) DETECTED
```

'RUMOR-MILL'

This is the first of a new feature called 'Rumor Mill'. Not 100% accurate but we believe you will find, over a period of time, that we will have our rumors fairly straight.

From SWTPC

It is a fact that there have been some big changes in SWTPC country. First the 6800 machine is replaced by the 'Bomb' the 6809. As everyone knows the 6809 is the 'Big Fist' company's offering to the 16 bit crowd. Our reaction to the 09 is enthusiastic. It brings a whole new realm of power to small computers. However; I feel that the 6800 has more power than any of us have fully used. I suggest that the 6800 will be around a long time for a lot of users. Wimper..you 80 types, the big dogs are in our yard, for a long time to come.

All this means that the days of everyone building kits is fast closing. The majority are being ordered, tailor made. Kind of a sad thing in a way; we sure learned a lot, the hard way. The kits will still be available from SWTPC but seems not many want to do like a lot of us did. Well...such is progress?

SWTPC is now delivering the 6809, so you that are getting the hang of the thing, let us know what you have learned and done. The new 'Winchester' with Century Marksman drives should be delivered soon. 20 meg-a-bytes, a lot of empty space to fill up. We look forward to it as our mailing list is sure growing.

GIMIX

Richard Don informs us that GIMIX is busy getting patches for all types of software, that is currently available for the 6800. These are to run under GMXBUG Ver. 3.0, and the 'Super Video Board' that thinks it is the whole machine. We have a review coming on this soon. The ability to create easily your own special character set and use it immediately, called as a program, is top drawer.

Richard has promised to keep us fully up-todate on all new and existing patches and overlays. We will publish them as they are received.

The list is something like this, as of this writing: SWTPC BASIC Ver. 2.0, Ver. 2.2, Ver. 2.3, Disk Ver. 3.0, FLEX Ver. 1.0, FLEX Ver. 2.0, PRINT.SYS, TSC Text Editor (FLEX Ver. 1.0), TSC Text Processor (FLEX Ver. 1.0), TSC Assembler (FLEX Ver. 1.0), TSC Text Editor (Tape), TSC Text Processor (Tape), TSC Assembler (Tape), TSC Micro BASIC PLUS Ver. 2.1, TSC Space Voyage (also Random Generator), TSC Relocator, TSC DeBug package, SSB Super Editor, SSB Super Assembler, GRT G2 BASIC 1.0, PERCOM miniDOS 1.2, PERCOM Touch Up for TSC Editor, PERCOM Assembler, TSC BASIC both tape and disk versions. Also, TSC BASIC F2, TSC BASIC D and HEMENWAY ASSOC. 6800/6809 Cross Assembler.

SSB

Watch for a lot of new Smoke software. We have not received anything as of this writing, but we understand that it satisfies the needs of those looking for 'business' type coding. We hope to have a rundown by next issue..so watch this column.

COMPUTERWARE

Computerware has developed a new 'small business' package to run under the SSB DOS. This package is to include Accounts Receivable, Accounts Payable, General Ledger (single entry type), Inventory and Payroll. I am told that the entire package will be well under $500.00. We know a lot of users just waiting for this type software.

There is a new version of their BASIC with some needed additions, such as 'PRINT USING' and 'ON ERROR'. This should be available soon. Their INVENTORY and MAIL LIST packages have been polished up and are now available.

PERCOM

Harold Mauch of Percom informs us of a lot of new products and software now available. There is a new 8" Disk System, a new Business package and new utilities for INDEX (their interrupt driven Disk Operating System). Also new is the 'FINDER' a program running in PERCOM SUPER BASIC, that is a general data manager.

I am also informed that there are many patches and overlays to existing software (and hardware) to ease the transistion from tape to disk, now available for immediate delivery. Running also on the PERCOM systems is the entire 6800 series of HEMENWAY ASSOCIATES STRUBAL+ compiler and support software. The PERCOM MAILING LIST program is reported to fill a need for those needing a complete package. Look for the new PERCOM VIDEO BOARD to be available real soon.

THOMAS INSTRUMENTATION

The new low cost SS-50 SUPER CPU board is now out and running. It runs on any S50 bus, so we are told. A review of this is in the mill, so keep watching.

SOUTH EAST MEDIA

SOUTH EAST MEDIA announced that they now have all sizes of disk available at great savings. The 5" disk are still $3.09 each (soft sectored, 10 and 16 hole hard sector) and the 8" soft sector (single side) is only $4.29 each. These are available in lots of 10 or more. Customer should include enough for shipping. Other types are available, so they report.

WORLDWIDE ELECTRONICS

Rumor has it that the IBM terminal they now sell will have a new and improved ASCII conversion board soon. The price is unknown but should be among the lowest priced selectric with ASCII advertised. It is supposed to do away with some of the IBM protocol that is now required on the present machine.

Well, that is about all for now. As we hear it we will let you know. And to you makers and sellers; if you have something new or juicy that we should know about, let me know.

* CLASSIFIED *

One SWTPC CT-64 Video Terminal, excellent condition, with all documentation...$200.00..firm, working fine. contact Don at 68 Micro Journal.

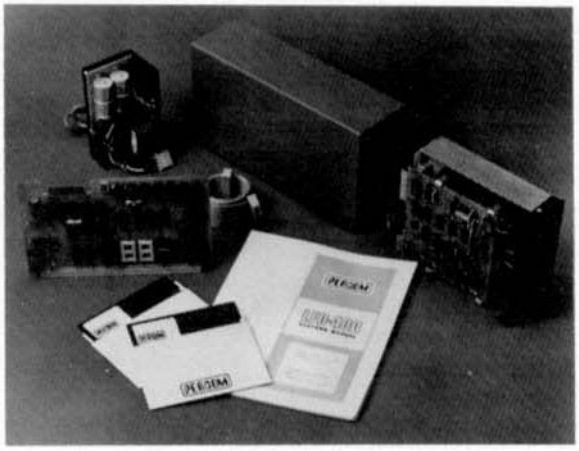
SWTPC 4K memory boards, 2102 low power chips, contact: Charlie Seagroves, AC 615 698-5002.

PERCOM LFD-400 Disk (A Review)

68 Micro Lab

The Percom LFD-400 disk system is the subject of this month's equipment review. The system used for this review contains two Pertec FD-200 5" disk drives. Although not double sided they are able to read and write both sides of the disk by the operator removing the disk and turning it over, then using the other side. A real savings in disk cost and additional data storage. We are informed by the Percom folks that all systems are shipped with either the Pertec or Wangco model-82 drives. Both of these drives are able to have the disk flipped over, and use both sides. It is our opinion that the Wangco drives are preferred to the Pertec drives because of easier disk insertion. However; either make of drive is perferred over the Shugart, which only uses one side. Some claim that using both sides is not good, due to the cleansing action of the jacket lining. We have used the double side method for over two years and have not experienced any known problems, due to reverse rotation. It is best that you consider the above when storing important data.

The LFD-400 is housed in an attractive blue steel case, the power supplys are enclosed in the rear, each drive has it's own power supply. In event of power supply failure it is highly unlikely that



you would lose the use of the entire disk system. It could still operate with the good power supply on the system drive (provided you have more than one drive on the system). Operating on only one drive, while not as efficient as two or more, allows continued operation.

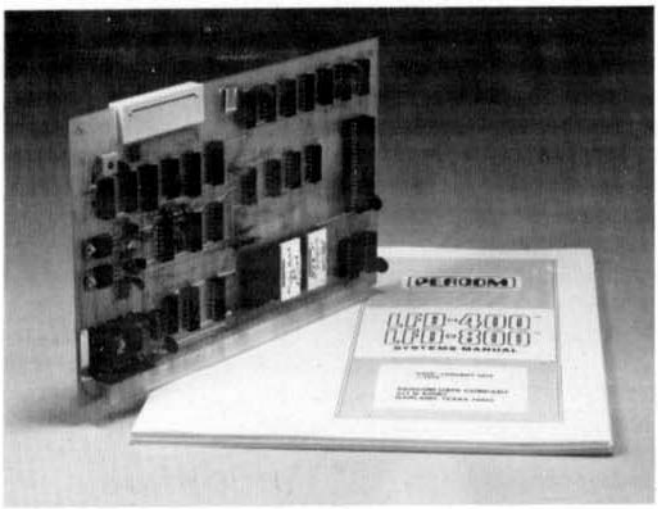
We will look at the various portions of the Percom LFD-400 disk system. Starting first with the power supplys and ending with the documentation. We will only cover the two disk operating systems furnished with each disk system. Furnished on EPROM (2708). A later review will be devoted to 'INDEX'. Furnished on disk only. INDEX is an interupt driven disk operating system that is very extensive. Index is available separately, from Percom, and is not completely compatible with the regular Percom disk operating systems and disk drivers. However, INDEX does operate with the Percom disk systems using two or more drives.

Power Supplys

It should be noted that the entire disk system, including the power supplys comes completely assembled. All that was necessary was to mount the drives in place, install the controller board on our S50 bus and bring the system up. The specs for the power supplys are rated as follows:
Plus 5v DC at 1.0 amps and plus 12v at 1.0 amps, rated continuous. The 12v surge current is rated at 1.8 amps. Operating temperature ranges from 0 degrees C to 50 degrees C. We have run the Percom dual system for days without removing power and the power supplys run well within the ratings.

The Controller Board

The disk drive controller board plugs directly onto the S50 bus. The flat ribbon cable from the drives is attached with card-edge gold plated connectors. Special low drop regulators are used on the controller board. This is especially welcome if your 12v supply is slightly weak or you have the slots of your motherboard loaded, causing the 12v line to run at a marginal level. Three EPROM sockets are provided for the disk operating system (supplied) using 2708 EPROM. The EPROM space is so arranged that additional programs may be burned and stored. The EPROM address starts at $C000. The controller board uses a 'proprietary adaptive data separator' correcting for 'bit shifting'. Bit shifting is a phenomenon observed when data is tightly recorded and appears to move when influenced by rapid flux reversals, as when writing to the disk.



The controller board allows the drives to 'spin' for approximately 4 seconds, if they are not being read or written to. This increases the life of the disk drives (the most expensive part of the entire system) by allowing them to run only when called to read or write. The controller board is 'buffered' on outputs to minimize bus loading. The quality and construction of the controller board is excellent and performed well, as expected.

MINIDOS
The Disk Operating System

MINIDOS was the original 'disk operating system' furnished by Percom with their disk systems. It is burned into a 2708 EPROM. It is backed up by an extension disk operating system, also in EPROM named MINIDOS-PLUSX. We will discuss them separately.

MINIDOS is a 'primitive' disk system. It is simple to use but does require external records to be maintained (program locations on the disk), to be explained later. If MINIDOS-PLUSX is installed, and should be as it is furnished at no additional cost, MINIDOS passes control to MINIDOS-PLUSX.

MINIDOS does not load or save between computer memory and the disk by 'program name' calls. This is where the extra effort comes in. For example; to load a program from the disk to memory the following must be entered at the console:

```
L 1025 FFFF
```

'L' is the single letter command for LOAD. 1025 contains the following; the numeral 1 indicated which drive is to be used. Drives are numbered 1 thru 3 (the system supports up to 3 drives). The 025 indicates the beginning block (or sector) to load to memory. The ending block is taken care of by the disk operating system (DOS). The disk has 256 byte sectors and stores approximately 89.000 bytes of data. There are a total of 350 blocks per disk (each side), numbered from 000-349. The controller requires 10 hole, hard sector minidisk. The $FFFF indicates that the program should be stored in memory at the same address as when it was saved. If the program is relocatable and you desire it loaded at another block of memory, then the $FFFF would be replaced with the address desired. If a starting address was saved then the 'program counter' $A048 and $A049 will be set to the proper address. The program is started, from the monitor, by typing 'G'.

To SAVE a program from computer memory to disk the following is typed at the console:

```
S 0100 1FFF 0100 1025
```

The 'S' indicates a save function. The $0100 is the beginning address of the program or data to be saved. The $1FFF is the ending address. The next $0100 is the 'transfer' or starting address. The 1025 indicates drive number 1 and block (or sector) 025 as the first sector to use on the disk. When saved the computer will respond with "LAST SECTOR=XXX", where XXX

is the last sector used. The instruction manual suggests that you "Write the last sector number on the storage jacket of the diskette together with the program name and starting sector for future reference.". MINIDOS supports error reporting.

While limited in the utility support normally expected or required by some users, MINIDOS is simple to use and gives the user access directly to the disk drivers. Also it uses only a small amount of computer memory; from $0000 to $001F, and some from $A000 thru $A07F (CPU stacking and storing area). The limitations need be of little concern as MINIDOS-PLUSX is also furnished making the presence of MINIDOS 'transparent'.

MINIDOS-PLUSX

MINIDOS-PLUSX is contained in an additional 2708 EPROM, which is installed on the controller board. MINIDOS is so configured that if the EPROM containing MINIDOS-PLUSX is installed it jumps to an address in MINIDOS-PLUSX; thereby making MINIDOS 'transparent'. MINODOS-PLUSX has many features that compare with any disk operating system we have used. The major drawback to this system is that it is restricted to 31 'named files'. File names consist of 6 characters with extensions. MINIDOS-PLUSX is available in three configurations.

1. On 2708 EPROM with RAM buffers at $A080-$A2AF. This is the configuration we tested.
2. On 2708 EPROM with RAM buffers at $7080-$72AF. This allows those not desiring or having memory beyond $A07F, to use low RAM space. (Complete information is given for modifing various memory boards)
3. On a disk containing the system for loading at either of the areas in memory. The disk contains complete source listings and is compatible with the Percom modified TSC Editor and Assembler.

Commands supported by this system are as follows: 'I' Initialize the disk (any drive you designate). 'S' Save from memory to disk, example: S NAME 100 1234 200. The 'S' implies SAVE and the 100 (hex) is the beginning address, 1234 (hex) is the ending address and 200 (hex) is the transfer address or program counter pointer. 'A' allocates additional space to any program or data file you might desire to expand. Expansion is accomplished in 10 sector blocks and may be repeated on any file. 'L' is for LOAD and is quite simple to use, merely typing: L BASIC would load BASIC to memory, providing you have basic on that disk. Typing 'BASIC' (no 'L') would load and start BASIC running. Should the Directory be 'bombed' it is still possible to get your programs off the disk by using the load by drive-track-and sector, as discussed earlier. 'F' is the command for the 'FILES' of the Directory. Typing 'F' causes the Directory to be listed to the console screen. The following information is displayed for each file on the disk. File name, first block (sector) used,last block used, beginning address, ending address and transfer address or program start address. All list information is in hex. If the program start is $FFFF then no transfer address is implied and automatic start is inhibited. 'R' RENAME renames files on the disk. A disk may have two or more files on a disk with the same name and extension. Only the first encountered will be used by the system. Care should be exercised in naming files. The Rename command also allows for file protection. 'D' DELETE deletes files from the disk. 'J' JUMP allows the user to jump to any address in memory. Example: 'J XXXX'. 'X' EXIT to the monitor. 'M' exits to MINIDOS. 'BACKUP' Backup copies data and files from one drive to another. 'COPY' copies individual programs from one disk to another. 'CREATE' Creates a file of the size specified. 'PACK' Packs or reclaims unused space on the disk created by deleted files. MINIDOS-PLUSX uses the error reporting function of MINIDOS plus and extends the reporting function with additional error codes.

An excellent 'money-saver' is the many 'USER GROUP' overlay and utility programs available from Percom to allow the user to upgrade his existing TSC, SWTPC and SSB tape software, for disk utilization. This eliminates the need to purchase the same software for disk operation. The patches to BASIC (BAND-AID) allow for data files as well as source file storage. Data files in the modified BASIC may be handled in random or sequence format.

Available also are additional overlays for various text editors and processors, as well as assemblers. These result in some

cases with programs with more utility than in the original form. These routines are called 'TOUCH-UP', and are also available from Percom. The list of additional utilites and software patches are too many in number to cover in this review, and are being added frequently. It is suggested that interested computer users contact Percom direct for more complete information.

In MINIDOS-PLUSX are utilities and programs that allow the recovery of 'bombed' disk. We have found them very handy and on one occasion essential. 'DISKEDIT' is the primary recovery program. It is very complete and works as follows: Because program location is known (disk location, track and sector), on the disk, it is possible to call from the disk, one sector at a time. This sector then may be modified or corrected in RAM buffer and returned to the disk - same track and sector. By this method much otherwise lost data can be recovered. We were able to reconstruct over half of a disk directory that had been bombed by a power 'flick'. When finished we could not tell that the disk had ever been tampered with. Needless to say a most important utility, and one that makes any disk system a joy to use.

Documentation

The documentation for the system is impressive. both in content and quality. Most routines are explained in detail and are accompanied by assembled source listings. We gave the system to 'novice' operators in the 68 Micro Lab and turned them loose with it. They all agreed that the documentation allowed them to become comfortable, in a short time, with the entire system, both hardware and software wise. We found no 'funny' surprises in the system or software.
We only experienced one failure for the entire period we have used the LFD-400 system. This was no fault of Percom. An IC on a Pertec drive controller board (supplied by Pertec) became thermal sensitive and caused some weird happenings for a short while. We were able to trackdown the offending component in about 20 minutes and Percom replaced the drive immediately. We could have replaced the IC inhouse but did not want to void any warranty. It is a good policy not to 'fiddle' with items in warranty. The system is guaranteed for 90 days.

Rating

A 68 Micro Journal lab rating of 'AAA' for the disk system. The EPROM software is rated 'AA'. The extended utility of MINIDOS-PLUSX upgrades the software system considerably. As stated earlier MINIDOS-PLUSX is now standard on this system, and is supplied with all Percom disk systems at no additional cost.

Additional information, concerning the Percom Disk Systems or software available may be had by writing:

PERCOM Data Corp
211 N. Kirby
Garland, Texas 75042
A/C 214 272-3421
or 1-800-527-1592

Rating scale:
AAA - Excellent
AA - Good
A - Fair (could be better but works)
P - Poor and may not always work properly
X - Not recommended for children (or anything else)

NEW PRODUCTS

Jerry Koppel of AAA Chicago Computer Center announced that they have modified their Editor to accommodate all their customers using the GIMIX-BUG Version 3.0 with the new 24X80 controllable video board.

He noted that the revised Editor supports both FLEX (tm) and SSB (tm) disk systems.

Additional information can be secured by writing:
AAA Chicago Computer Center, 3007 1/2 W. Waveland Avenue, Chicago, IL 60618, or call A/C 312 459-0450.

IMPROVED BOOT FOR SWTPC MINI-DISK SYSTEM

Anyone who has used the SWTPC mini-disk with the SWTBUG ROM has probably experienced unreliable disk booting. This problem is a result of two shortcomings in the implementation of the 'boot' in SWTBUG at $E28F. One flaw is that it does not retry in the event a CRC error is detected. The second defect is that it does a timed delay rather than test the status resister ($8018 bit 7) for a NOT READY condition before issuing a RESTORE command to the FD1771 disk controller. This can result in the RESTORE command being ignored if the chip is not ready.

The FLEX 1.0 user's guide page 3.4 offers a recommended boot that does a retry if a CRC error is detected; however, it too does not test the NOT READY bit.

The correction is to simply monitor the status register to insure the NOT READY flag is cleared before issuing a RESTORE command. This may be achieved as follows...

REPLACE

```
0107 CE 0000      LDX   0
010A 08      OVR  INX
010B 09           DEX
010C 09           DEX
010D 26 FB        BNE   OVR
```

WITH

```
0107 F6 8018 LODP LDA B COMREG READY?
017A 2B FB        BMI   LOOP
```

Firmware implementation may be slightly more difficult unless your system provides space for additional EPROM. In my system I placed the improved boot in EPROM external to SWTBUG and changed the jump address in the SWTBUG command table to reflect the location of the improved boot.

It's great to hack with a system that boots reliably!!!!!!

Allen Clark
2502 Resal Oaks Lane
Lutz, Fla. 33549

```
8018                COMREG  EQU     $8018
8014                DRVREG  EQU     $8014
801A                SECREG  EQU     $801A
801B                DATREG  EQU     $801B


0100                        ORG     $0100


0100 B6 80 18   START   LDA A  COMREG
0103 4F                 CLR A
0104 B7 80 14           STA A  DRVREG
0107 F6 80 18   LOOP    LDA B  COMREG
010A 2B FB              BMI    LOOP
010C C6 0F              LDA B  #$0F
010E F7 80 18           STA B  COMREG
0111 8D 31              BSR    RETURN
0113 F6 80 18   LOOP1   LDA B  COMREG
0116 C5 01              BIT B  #1
0118 26 F9              BNE    LOOP1
011A 7F 80 1A           CLR    SECREG
011D 8D 25              BSR    RETURN
011F C6 9C              LDA B  #$9C
0121 F7 80 18           STA B  COMREG
0124 8D 1E              BSR    RETURN
0126 CE 24 00           LDX    #$2400
0129 C5 02      LOOP2   BIT B  #2
012B 27 06              BEQ    LOOP3
012D B6 80 1B           LDA A  DATREG
0130 A7 00              STA A  0,X
0132 08                 INX
0133 F6 80 18   LOOP3   LDA B  COMREG
0136 C5 01              BIT B  #1
0138 26 EF              BNE    LOOP2
013A F6 80 18           LDA B  COMREG
013D C4 0C              AND B  #$0C
013F 26 BF              BNE    START
0141 7E 24 00           JMP    $2400
0144 20 00      RETURN  BRA    RTN
0146 39         RTN     RTS
                        END    START

NO ERROR(S) DETECTED

 SYMBOL TABLE:

COMREG 8018   DATREG 801B   DRVREG
LOOP2  0129   LOOP3  0133   RETURN
START  0100

8014   LOOP   0107   LOOP1  0113
0144   RTN    0146   SECREG 801A
```

Al Tejera
3904 Hunt Rd.
Apt. #225
Tampa, Fla. 33614

STRUBAL+:
BASIC plus Structured Programming
yields a powerful language

Robert D. Grappel
Hemenway Assoc.
Boston, MA 02108

STRUBAL + A COMPILER

The design of a computer language deals with a wide range of tradeoffs. A new language is created so that programmers can write programs more easily, faster, more efficiently than is possible with existing languages. On the other hand, there is a definite benefit to retaining the elements of other languages with which potential users feel familiar. The language should provide great flexibility, allowing the programmer range to accomplish the task, yet not clutter the language with a myriad of special cases and complex rules. There is the tradeoff of speed versus storage; programs should execute fast but not require excessive memory or other support. These issues become especially important in dealing with micros since they are limited both in power of instruction set and in peripheral power. One other important point, programs should be easy to read. Clarity of the source program is vital to understanding and debugging any language.

BASIC forms the basis

The BASIC language is one of the most popular computer languages in use today; certainly it ranks as the most commonly available language for micros. BASIC is versatile and easy to learn; a large base of programs and trained programmers exist. For this reason, most of the BASIC statement types appear in STRUBAL+. This allows a BASIC programmer to use STRUBAL+ immediately--avoiding the long learning curve changing languages often produces. There are some significant differences between BASIC and STRUBAL+, however. BASIC is usually implemented as an interactive interpreter: an implementation that exacts penalties in program-execution speed and limits direct access to the computer. It would be hard to write operating systems or interrupt processes in BASIC. For these reasons, STRUBAL+ is a compiler. It generates object code for the computer. This has benefits:

1. much faster execution of programs
2. typically smaller programs

but there are also drawbacks:

1. debugging at machine level
2. multiple-step procedure to compile programs and run

Another major difference between BASIC and STRUBAL+ is the elimination of line numbers. It is not immediately obvious that the BASIC statement

```
100    GOSUB 200
```

directs the program to a routine that computes the average of a set of numbers. Contrast this with the STRUBAL+ statement

```
COMPUTE GOSUB AVERAGE
```

The use of descriptive names (labels) for program lines adds a great deal to the readability of programs. Labels are used at the discretion of the programmer; only those lines which are to be transfered to must be labeled.

Data types add efficiency

Most BASIC implementations maintain all values as floating-point numbers. Floating-point is a very inefficient format for micros, requiring extensive software. Micros work best with integers, either 8 or 16 bit values. Many programs do not need the range provided by floating-point; it is important to give the programmer the ability to define values as floating-point or integer as required. Thus, the STRUBAL+ compiler uses fast integer arithmetic with integer values, only coding in floating-point when the user requires it. This requires the use of two statement types:

INTEGER	declare variables integer (16-bit)
DEFINE	declare variables floating-point

The compiler's integers are 16-bit 2's-complement values; its floating-point values are BCD-encoded mantissas with two digit BCD exponents. The compiler allows a choice of the precision of the floating-point format within the program. Values may be stored with as many as 14 digits or as few as 4 digits.

The use of fewer digits results in higher speed, furnishing the option of trading accuracy for speed as desired. Business applications, which require extended accuracy, can have it at the expense of execution speed. The floating-point precision is selected by the DIGSET statement, in the format

DIGSET=integer value (could be an expression)

Programs default to 14 digits and always allow room for all the digits in floating-point values. The scientific functions supplied with STRUBAL+, however, actually work to only about 8 digits of precision.

Programs do not deal only with numeric quantities; frequently they must work with character data. STRUBAL+ provides character-manipulation facilities that allow character-string operations similar to those in many BASIC implementations. Character strings are defined by the "DSTRING" statement. A string's maximum length must be given in the statement as shown below, where INAME is defined as having a maximum of 72 characters.

```
DSTRING INAME(72)
```

Actual string length may vary as the program runs so long as the maximum length is not exceeded. Character strings may be input and output. The percent-sign (%) (used as a prefix) indicates to the compiler that data is to be handled as a string for I/O. A set of STRUBAL+ keywords manipulate strings.

CONVERT convert a numeric value into a string

LENGTH get the present length of a string

STRING performs four types of string operations

1. Assign a fixed string value

```
STRING INAME='THIS IS A STRING'
```

2. Assign one string to another

```
STRING INAME=TEMP
```

3. Compare one string to another

```
STRING IF INAME .EQ. TEMP THEN SKIP
```

4. Concatenate strings into one

```
STRING INAME=TEMP,'AND','THIS TOO'
```

VALUE convert a character string into a number

XTRACT extract a substring from a given string

Structuring your programming

One of BASIC's major weaknesses is its lack of control structures. Although GOTO, GOSUB, the IF-statement and the FOR-NEXT exist, modern programming practice has shown that they are insufficient to provide easily-read and maintained programs. The use of labels instead of numbers helps here, but STRUBAL+ also adds statements which extend the range of program control operations available to the programmer.

One of STRUBAL+'s powerful control structures is the WHILE-BLOCK, which defines a program section that will be executed so long as an expression associated with the WHILE statement evaluates nonzero (STRUBAL+ assumes nonzero is true and zero is false). WHILE-BLOCKs may be nested and mixed with other control statements like GOTO and FOR-NEXT. An example of the use of the WHILE-BLOCK (shown below) also illustrates some of the file-I/O commands available in STRUBAL+.

The function .NOT. performs the logical negation of a value. (bitwise inversion) The function .EOF. indicates the occurance of an end-file during a file read. The READ statement takes in data from an open file.

```
WHILE .NOT. .EOF.
READ (FCB) %DATASTRING
PRINT /,%DATASTRING
BLOCK
```

This program fragment reads and prints the entire file. (Things like the slash [illegible] in the print statement and the use of FCB in the READ statement will be discussed later.)

STRUBAL+ has extended BASIC's IF statement to include some new formats. The

BASIC form:

```
IF expression line-number
```

is handled in STRUBAL+ by the syntax

```
IF expression THEN label
```

Many BASICs also allow a statement to replace the line number. STRUBAL+ allows the same syntax, as shown.

BASIC:

```
IF expression GOSUB line-number
```

STRUBAL+:

```
IF expression GOSUB label
```

STRUBAL+ also allows an ELSE clause on IF statements, furnishing a second transfer label when the expression evaluates false (nonzero).

```
      IF expression THEN label-1
    ELSE label-2
```

These forms of IF are also allowed on STRING IF statements.

Like some BASICs, STRUBAL+ has a form of computed-GOTO and GOSUB statement. This structure allows the value of an expression to determine the transfer point. In STRUBAL+ the computed version is a combination of the ON statement and a modified GOTO or GOSUB.

```
ON expression
GOTO (label-1, label-2, ...)

ON expression
GOSUB (label-1, label-2, ...)
```

Global, local, and Common variables

In BASIC programs, all variables are global--that is, they are accessible from any statement in the program. The same is true of STRUBAL+ variables when they are compiled together with the program using them. You can, however, compile program modules independently; the modules are then combined to form the total program. The variables will then be local to their modules unless you make special definitions.

Modules can communicate with each other via the CALL and PROCEDURE statements. The CALL statement is similar to the GOSUB in that it transfers to the named routine and returns to the calling program. CALL, however, refers to a transfer point in an independent module, not part of the calling module. Also, CALL defines a set of variables which will made available to the called module. The PROCEDURE statement is used in the CALLed module to define the variables local to the module which will be made equivalent to the variables in the CALL statement.

```
* MAIN PROGRAM HERE
*
      INTEGER I,J
      CALL GETDAT(I,J)
      PRINT /,I,J
      STOP
      END

* PROCEDURE PROGRAM HERE
*
      PROCEDURE GETDAT(A,B)
      INTEGER A,B
      INPUT 'ENTER VALUE ONE',A
      INPUT /,'ENTER VALUE TWO',B
      RETURN
      END
```

It is also possible for variables to be defined global for all program modules in the total program. This mechanism is called COMMON. The concept of COMMON is familiar to FORTRAN programmers; it means that an area of memory is set aside for a group of variables that all programs can reference by refering to the common memory area. STRUBAL+ supports a single common area; programs access it by offsets from its origin.

```
* MODULE ONE USING COMMON
*
          CSECT
          INTEGER I,J,K
          DSTRING INAME(72)
          CEND


* MODULE TWO ALSO USES COMMON
*
          CSECT
          INTEGER A,B,C
          CEND
```

The STRUBAL+ statements CSECT and CEND

tell the compiler that the variable definitions between them are to reference the common space. Hence, in the example, I,J, and K in module one are the same memory locations as A, B, and C in module two.

Assembler works with STRUBAL+

At times, programming in assembly language is the best solution to a given problem. Some BASIC implementations provide mechanisms to provide the power of assembly language. PEEK and POKE statements do some of the work, both in BASIC and in STRUBAL+. BASIC provides the USER function to call assembly subroutines, but this is clumsy at best. A compiler, alternatively, can handle assembly statements directly because the compiler contains a complete assembler. Hence, STRUBAL+ was designed to allow the programmer to imbed assembly language wherever desired—right in the middle of a program. Imbeding code in this manner is termed "crutch coding"; there are times when it proves very useful.

The compiler makes all CPU registers available to the programmer. Crutch-code can access any variable or label by name. In addition, special statements and syntax aid in the use of crutch-coding:

1. Hexadecimal integer constants

2. GOSUB @expression
GOTO @expression
(the expression's integer value becomes the transfer address)

3. GETAD statement (get the address of a variable as an integer)

Note that one assembly feature has already been illustrated: the use of an asterisk (*) to indicate comment lines. The STRUBAL+ compiler generates no code at all for comment lines. (Unlike BASIC where REM statements take up memory) This encourages abundantly-commented programs.

Formating features aid I/O

One weakness of many programming languages, especially on micros, is their inability to easily produce formatted output. STRUBAL+ provides strong support in this area. You can insert format specifications into any I/O statement. (INPUT, READ, PRINT, etc.) For numeric values, the format may specify the number of places left and right of the decimal point. You can output integers with two decimal places (useful if these are money quantities for instance), and floating-point values that are easily made to lie in columns. Strings are formatted to have a desired number of characters, and are padded with spaces or truncated as necessary. The following examples illustrate STRUBAL+ formating.

```
    PRINT /,[6],IVAL
*
* print IVAL as a 6-digit integer
*
    WRITE [8,2],X
*
* write X as 8-digit floating-point
* with 2 decimal places
*
    PRINT [20],$ISTRING
*
* print ISTRING
* as 20 characters
*
```

In addition, a spacing function, indicated by the greater-than sign (>)

```
    PRINT >I,'A CHARACTER STRING'
*
* indent I spaces and print '.....'
*
```

allows controllable spacing for plotting, forms, etc. The slash (/) indicates a carriage-return and linefeed. These features allow precise control over the format of data within STRUBAL+.

COBOL provides data structures

Beyond the data types already described, STRUBAL+ supports data structures which are in fact mixtures of types. This is done by defining "dummy-names" which address locations in a structure. The STRUBAL+ DSECT and DEND statements are used to make these definitions within what is called a "dummy section". As an example, the following defines a structure called RNDBUF which is a character string and which has 3 substrings called RNAME, RADDR, and RNUM.

```
        DSTRING RNDBUF(28)
*
```

```
* defines actual storage
*
       DSECT
       DSTRING RNAME(10),RADDR(10)
       DSTRING RNUM(5)
       DEND
*
* dummy names defined here
*
RNAME  EQUALS RNDBUF
RADDR  EQUALS RNDBUF+11
RNUM   EQUALS RNDBUF+23
*
* assign dummy names to locations
*
```

Now the dummy names address locations within the structure. They may be assigned values, printed, etc.

```
       STRING RNAME='BOB GRAPPEL'
       STRING RADDR='101 TREMONT ST.'
       PRINT /,$RNAME,/,$RADDR
```

The structure need not be homogenous; consider the following example which not only builds a structure containing all the STRUBAL+ data types but also assigns initial values to them.

```
       DSECT
       INTEGER I,J
       DEFINE X
       DSTRING NAME(10)
       DEND
*
I      FDB 1
*
* assign I the value 1 as
* a 16-bit integer
*
J      FDB 50
*
* assign J the value 50
*
X      FFP 10.55
* assign X the value 10.55
* as floating-point
*
NAME   FCC 'DATA'
*
* assign NAME the value DATA
*
```

Note the special statement FFP which builds a floating-point constant. Note also that FDB and FCC are actually assembler pseudo-instructions which are crutch-coded here.

Recursive programming can help

STRUBAL+ is not a recursive language by design. It does not automatically generate re-entrant code. Therefore, for those occasions when it is necessary to make STRUBAL+ programs work recursively, two new statements, PUSH and POP, were added. These statements allow any variable to be pushed and popped on the system stack. The programmer is responsible for stacking the necessary variables and for removing the variables in the proper order (reverse of the stacking order).

File control for sequential files

STRUBAL+ provides support for sequential disk files through calls to the underlying DOS. Each file is assigned an area of memory called the "file-control block" or FCB. The programmer is responsible for supplying an FCB of sufficient size for each file which will be open. The FCB name is supplied in each statement which manipulates the file. The STRUBAL+ statements which work with files are:

```
OPEN (FCBNAME)       open a file
CLOSE (FCBNAME)      close a file
READ (FCBNAME)       read file data
WRITE (FCBNAME)      write to a file
REWIND (FCBNAME)     file rewind
```

In addition, two file functions are supplied:

```
.EOF.  end-file found during read
.ERR.  I/O error occurred
```

One system function to initialize the DOS is supplied. It is called INITIO. Sequential files are opened with filenames as specified by the DOS supporting them. STRUBAL+ requires that ';I' be appended to the name of a file to be opened for input and ';O' be appended to the name of a file opened for output. The example illustrates how to read, write, and print a file.

```
    DSTRING LINE(80),IFNAM(30)
    DSTRING OFNAM(30)
    DSTRING IFCB(166),OFCB(166)
*
* this system uses 166-byte FCBs
*
    CALL INITIO
    INPUT /,'INPUT FILE?',$IFNAM,/
    STRING IFNAM=IFNAM,';I'
```

```
*
* now have input filename
*
    INPUT /,'OUTPUT FILE?',%OFNAM,/
    STRING OFNAM=OFNAM,';O'
*
* now open both files
*
    OPEN (IFCB) IFNAM
    IF .ERR. THEN INERR
*
    OPEN (OFCB) OFNAM
    IF .ERR. THEN OTERR
*
LOOP   READ (IFCB) %LINE,/
    PRINT [80],%LINE
    WRITE (OFCB) [80],%LINE,/
    IF .NOT. .EOF. THEN LOOP
*
    CLOSE (IFCB)
    CLOSE (OFCB)
*
    STOP
*
INERR  PRINT 'INPUT ERROR',/
    STOP
*
OTERR  PRINT 'OUTPUT ERROR',/
    STOP
*
    END
```

The slash (/) in the WRITE statement causes a record mark (linefeed) to be written into the file. The slash in the READ statement causes the file to be read up to the next record mark. This provides a fast way of skipping through a sequential file.

Putting the "plus" in STRUBAL+

One additional feature which makes STRUBAL+ more powerful than competing languages is the ability to write macros or functions. STRUBAL+ grew out of a macro-assembler, so these facilities were natural additions to STRUBAL+ syntax. A macro or function is effectively a shorthand way of writing complex programs as a set of progressively defined new instructions. These new instructions are user-defined names attached to a defined set of STRUBAL+ statements. These statements may include parameters which change each time the set is invoked.

As a simple example, consider the task of generating a pause in the execution of a program. STRUBAL+ provides the DELAY statement for this, but DELAY can generate at most a 65.536 second pause. (DELAY takes an integer argument in milliseconds.) A STRUBAL+ FUNCTION (like assembler MACRO) could be defined as follows:

```
      FUNCTION WAIT
*
* the instruction's name is WAIT
*
      FOR I=1 TO &1
*
* &1 indicates a passed parameter
*
      DELAY 60000
*
* DELAY one minute per loop
*
      NEXT I
      FEND
```

Invoke the function by name
with the desired parameters.

```
      WAIT 10
*
* a ten minute pause
```

STRUBAL+ provides additional features for working with macros and functions. The zeroth argument (&0) is set to the number of parameters in the given invokation. The compiler can generate labels automatically within the body of the macro/function so that there will not be duplicate labels in the program module. Libraries of macro/function definitions can be included in the compilation of a program module. Conditional compilation instructions are available. With functions and macros, the programmer can produce all sorts of new capabilities within STRUBAL+, effectively customizing it into a new language.

Using the STRUBAL+ compiler

The process of compiling a program takes several steps. The compiler first converts the source program into assembly language in a single pass. This intermediate file is saved on disk for later passes. The compilation can produce a source listing, with or without the intermediate assembly language. An alphabetically-sorted symbol table may be printed. The choice of assembly language as the intermediate product makes it easy

8300

SWITCH-INDICATOR CONTROLS

External Switches	Power On-Off
	Select-Deselect
	Line Feed
Internal Selector Switches	Print Direction (↕ or ↔)
	SO/SI or SO only
	Non-Auto LF or Auto LF on CR code
	Non-Printing or Printing on LF, VT, FF codes
Internal Switches	Paper Empty
	Case Cover Lock

FEATURES

Standard

- Bidirectional Printing
- Character Set of 96 Symbols
- Tractor Feed
- One Line Internal Buffer
- 80 Character Print Line
- Double Size Character Set
- Low Cost

SPECIFICATIONS

Physical

Height	7.3 inches
Width	17.7 inches
Depth	14.8 inches
Weight	22 pounds

Environmental

Temperature	-25°–60°C (storage)
	10°–35°C (operating)
Relative Humidity	0–90% (storage)
	10–80% (operating)

Power Requirements

Voltage	115VAC±10%, 60Hz
Watts	100W operating, 7W stand-by

CHARACTERISTICS

8300

Print Features	125 characters per second
	60 lines per minute
	8.0 inches printable width
	10 columns per inch (normal width)
	5 columns per inch (double width)
	1/6 inch line spacing
Form Feed	Pin Feed method
	10 lines per second (slew speed)
	Loading from either bottom or rear
Forms	Pin-feed type
	4.5 inch–9.5 inch including sprocket margins
	0.013 inch maximum form thickness
Interface	8-bit parallel method
Control Signals	ACKNOWLEDGE, BUSY, SELECT, DATA STROBE, INPUT PRIME, FAULT, INPUT BUSY, PAPER EMPTY
Control Codes (ASCII)	CR, LF, VT, FF, CAN, SO, SI, DC1, DC3, GS, RS, US
Character Format	96 characters ASCII
	5 x 7 dot-matrix
	Impact printed in normal width and double width
Character Buffer	1 line (80 characters in normal width, 40 characters in double width
Print Head Life Expectancy	100 x 10^6 characters

SWTPC

SOUTHWEST TECHNICAL PRODUCTS CORPORATION
219 W. Rhapsody
San Antonio, Texas 78216 **(512) 344-0241**

6540 SERIAL PRINTER

SPECIFICATIONS

Physical

Height	10 inches
Width	27 inches
Depth	19 inches
Weight	85 pounds

Environmental

Temperature	32–104°F, 0–40°C
Relative Humidity	10–90%

Power Requirements

Voltage	115 ± 10% AC, 60Hz
Phase	Single
Watts	150

FEATURES

Standard

Bidirectional Printing
Horizontal and Vertical Tabs
Character Set of 96 Symbols
Character View
Forward and Reverse Line Feeding
Line Feeding in Increments of 1/2, 1/6 and full line
512 Character Internal Buffer
132 Character Print Line
Double Size Character Set

SWITCH-INDICATOR CONTROLS

On/Off, On/Off Indicator, Run/Hold, Home Paper, Forms Override, Space Paper, Vertical Positioning, Data Communications: Local/Remote, Half/Full/Echoplex, 110/300/1200 baud, Ready Indicator, Auto Answer.

CHARACTERISTICS

6540

Print Method	Serial/Impact
Character Structure	9 x 7 Dot Matrix (.105" high x .075" wide)
Printing Speeds	
Print Rate	120 characters per second
Tab/Carriage Return Equivalent Rate	36 inches per second 165 characters per second unidirectional
Line Feed	4.5 inches per second
Data Input	Serial
Code	USASCII
Format	
Print Positions per Line	132
Horizontal Spacing	10 characters per inch
Vertical Spacing	6 lines per inch
Forms	
Dimensions	2½" to 15" width
Type	Continuous, sprocket fed
Number of Parts	Original and 4 carbons
Inking System	Cartridge ribbon
Transmission Rate	110, 300, 1200 baud—Operator selectable
Interface	RS 232-C or 20MA current loop
Type	Asynchronous

SWTPC

SOUTHWEST TECHNICAL PRODUCTS CORPORATION
219 W. Rhapsody
San Antonio, Texas 78216 **(512) 344-0241**

to read what the compiler has produced. It is also easy to edit the assembler file to optimize the code generated by STRUBAL+.

The assembler within STRUBAL+ is a two-pass macro-assembler which produces a relocatable object module. The assembly may produce listings and symbol tables too. The object module is saved on disk for processing by the Linkage Editor program. Both the compile and assemble phases may produce a cross-reference file on disk instead of an assembly or object file. This cross-reference file contains data which is processed through the XREF program to output a listing of every label in the program and each line on which it was referenced.

Relocating libraries add support

The code generated by the compiler makes use of several libraries of routines. The first library, RUNTIM, supports the basic system needs. (16-bit arithmetic and logic, I/O control, procedure calls, etc.) The second library, FLOT, provides basic floating-point support. The third library, SCIFUN, provides the scientific functions. (SIN, COS, LOG, etc.) A fourth library, DRIVER, supports disk I/O as a link to the system DOS. The RUNTIM library is always required; but the other libraries may be omitted if their functions are not needed. The process of linking all these libraries together with the user program modules is the task of the Linkage Editor. The library functions are also available to the assembly-language programmer.

Putting the pieces together

The STRUBAL+ compiler produces relocatable object modules. The system support libraries are also supplied as relocatable modules. To combine these into an executable program requires the use of a program called a Linkage Editor. The program relocates the modules to reside at any desired memory address. All addresses within the modules are automatically changed to the proper relocated positions. All references to external names (procedure calls, etc.) are resolved and the necessary addressing supplied.

The Linkage Editor is a rather complex STRUBAL+ program with 20 user commands and it is capable of varied operations. Programs can be linked to reside where no memory exists in the computer running the Linkage Editor. Multiple program segments and COMMON areas can be built. Linkage Editor can print load maps and module directories. It can also search through files for the required program modules it needs to resolve pending references. Thus, it is not always necessary to load an entire file if only one module in it is needed. The output of Linkage Editor is a disk file which contains the executable program.

PRO-CON (one side)

Tom Harmon
HHH Enterprises
Box 493
Laurel, MD 20810

THE CASE FOR LARGE DISK OPERATING SYSTEMS

A rebuttal to page 5, Vol 1, Issue 3 of the '68' Micro Journal. "The case for small disk operating systems."

By Rev. Thomas Harmon
President: H H H Enterprises
President: U L C of Laurel
President: Smoke Multi-users Group

Q: Both the locomotive & the burro carry freight, but which is the better choice for a narrow mountain trail?

A: Since May 1869 the railroads have been carrying freight over the mountains and the rest of the country.

I am against small disk operating systems for small computers. In fact, after reading some articles (or adds), I find a lot of mystery. I can not see the economics of using a small disk operating system that can not be expanded to use most of the existing software that is on the market.

All references to paragraphs are (ref) and they are to the article of issue number 3, page 5.

(ref par 7). Yes, most of us bought a micro to get away from the expense of the big computer. But most of us, I think, would not mind having the power (complexity) of

the big machine. The insinuation that a good disk operating system is expensive, I believe is untrue.

Mr. Mauch of Percom Data Company, claims that INDEX TM is good and only costs $99.95. That doesn't sound too expensive. Of course you can buy the Smoke Signal Broadcasting DOS for only $39.95, and it includes a disk basic that I believe is several versions ahead of Super Basic. But let us stay with economy.

Percom LFD-400 advertisements don't mention any power supplies. They also say that it comes with MINIDOS-PLUSX (and it says that you get the listings for MINIDOS?). Lets add up the prices for an LFD-400 and a Smoke ABFD-68.

LFD-400=$599.95, plus (if you want more then 31 named files), INDEX TM=$99.95, for a total of $699.90. The ABFD-68=$649.00 (more then $50 less), and includes that big disk operating system, (unlimited file names, true random access, free Basic [supports both tape and disk], patches for using your tape version of CORES for both tape and disk, and an huge amount of those needed 'disk transients'). And for less then $100 more then LFD-400 the Smoke comes with a box, power supplies and cables for three drives. As far as taking up a lot of memory, Smoke's DOS takes up 4K (sequential), or 6K (random). Please note that both are supplied and you can choose between them. Also they can be ordered at three (3) different address locations, (not forcing you to $A000). Let's also mention the 4 drives and the fact that the Smoke DOS doesn't care whether they are 5 or 8 inch, or single or double sided!! All programs ever written to run on ANY version, from the first to the last, will run on any new version. (random files will not run on older sequential DOS).

(ref col 2 par 5) Does your data transfer get slower with a larger DOS? Since the speed of data transfer from a floppy is SET by the speed of rotation (always 300 rpm for 5 inch and 360 rpm for 8 inch) and the format on the disk, the only difference in speed comes from the software (DOS). My Smoke DOS is not the fastest in the world. However most of the speed reduction (almost all), comes from error checking and recovery attempts when there was a flake of dust on your diskette. I certainly am happy to know that mine certainly makes a hell of a try at keeping my data clean. Now a real zinger: We at SMUG have used a Smoke system to read other soft sector diskettes. They include: IBM format, FLEX, CPM, Apple and others.

(ref col 2 par 8) Most Basics can be trioked into reading a sequntial file semi-randomly by using a for-next loop. Don't need anything special there. But how can you run PERCOM's basic programs on a standard Basic???? I don't think that a random file Basic (with put and get, and write to last record, etc.), can be faked.

(ref col 2 par 12) Is it a fact that systems programmers really think that THEDOS (Mr. Mauch's name for a substantial disk operating system) is ideal for everyone, and the systema programmers arguments for a complex DOS are self-serving????

By the way, Mr. Mauch says (ref col 2 par 11)," two types of users come to mind.". Where is the second user???

(ref col 2 par 13) "Like the model railroader, some computer hobbyists like to emulate large computing systems." This seems to say that some model railroaders don't like to emulate large systems (railroads). As president of the local model railroad club, I was under the impression that they ALWAYS emulated a large system.

(ref col 2 par 14) I can't think of a better use for memory then a GOOD disk operating system, after all the DOS is the foundation that you build your system on.

Please note that I find a large number of articles on how to fix your SWTPC disk system hardware, and how to fix your MINIFLEX TM or FLEX 1.0 or boot your FLEX 2.0 TM. and the only articles on the Smoke Signal Broadcasting hardware and software

are how to add new goodies to the system. I wonder what that means. How about those folks with MINIFLEX? How do they read those disks on FLEX 2.0? How about the folks with the PERCOM that are buying the Smoke MF-68-U to convert their hardware to be Smoke compatible, just to get at a real Basic compiler and FORTRAN and SPIRIT and PILOT and TEXT PROCESSING and LISP and a Motorola mnemonic compatible assembler and true random access Basic (inter. or compiler).

Now let's look at just one big system DOS goody. REPAIR, is a small program that will allow you to recover those diskettes that were blown away by operator error or otherwise. This is because the Smoke DOS has both foward AND REVERSE linkage pointers in each sectors. Still think you can get away with a 'fast cassette' type of miniDOS why not try JPC PRODUCTS CO. who has a real fast cassette system, with Named files, Provides a directory, has speed (4K in 8 seconds), Saves files from memory, Loads files from diak to memory, Handles 'linked files', Has 'Auto-run', is available at two (2) different address locations. In ending, I think that if you want a disk system, you must have a good (read complex) DOS. If not go get a tape system, it certainly is a lot less money then an LFD-400.

Another rebuttal:

In response to the letter of issue 3, page 16, I would like to relate our (SMUG) experience with the MSI DOS and that system.

The version of the DOS we (SMUG) had was earlier than the Software Dynamics operating system, and was very similar to MINIFLEX. That means the damn thing lived at low address in the machine and was always in the way or had to be rebooted from one program to another. It used a block mode of storage, (you have to repack the disk, often), there were errors in it, it had no transients to speak of, the seek time for the drives was not optimized, and WORST OF ALL, IT USED WOFTWARE TIMING LOOPS FOR CRITICAL TIMING!!! Just try to change the CPU speed!! Rather then rewrite the MSI DOS to operate properly (meaning getting best flexability), we just changed the boot and I/O routines of the Smoke to operate those hard sectored drives. (interesting, no?). I also know that I had a Smoke disk system running before I ever saw an FD-8 system operate! When was the last time anyone used RT-68 and multiuser programmes on an FD-8 or on a MINIFLEX ?

In closing (finally), let me just say that the above material may be self-serving as I am a Smoke dealer. However, I personally don't think that anyone out there can beat my system at any level, because I have a very good foundation in my large DOS to build on. I also have had to advise and repair many systems other than the Smoke, and they were always lesser in utility to me.

CON-PRO (The other side)

Harold Mauch
Percom Data Company
211 N. Kirby
Garland, TX 75042

After reading Tom Harmon's rebuttal to my essay in Issue #3, it is fairly obvious he has not used and is not familiar with the Percom Disk. Consequently his is hardly a fair and knowledgeable rebuttal.

My Grandpappy used to say, "you can't tell much about a horse unless you get on and ride."

While I own and am quite familiar with both the Smoke and FLEX disk systems I obviously have a strong bias but not without bases or reason. Incidently between the Smoke and FLEX systems I have a slight preference for the Smoke system primarily because it is more fully software supported than FLEX. Both systems are quite good, however neither system handles I/O in a manner I consider acceptable.

My principle argument for small disk operating systems stems from a desire for economy and simplicity. The original manscript submitted to the '68' Micro-Journal included a comparison chart between the Percom and Smoke systems. The chart was editorially omitted but is repeated

here to validate my argument for economy. The bottom line speaks for itself.

One of the early products from Percom was a simple cassette interface for the SWTP 6800 computer. Despite the obvious superiority of a disk system over cassettes, the little cassette interfaces continue to sell. The message I get from this is that there are a lot of 6800 computerists out there who are not willing to spend the Kilobuck for a disk. This observation is the reason for"the approach Percom has taken. We want to provide the hobbyist with a fast replacement for his cassette which can evolve as his finances permit into a fully blown disk operating system. The transient file and memory requirements of the Smoke, FLEX, CP-68, and INDEX operating systems simply do not permit this level of simplicity.

I am sure Mr. Harmon and I could engage in endless debate most of which would be of little value to the reader. However I would like to answer some of Mr. Harmon's specific questions and inferences:

a) The LFD-400 IS supplied with disk drive power supplies. Incidently the Percom power supply and enclosure is an excellent way to power and enclose the Smoke ABFD-68 at less cost than the Smoke power supply and enclosure.

b) The Percom User's Group diskettes contain the source for MINIDOS-PLUSX, patches for CORES TSC and Microware software, as well as numerous additional disk transient commands and utilities. Source files are included for those wishing to modify or improve the programs.

c) The Percom SUPER BASIC has PRINT USING and DOS access features which do not exist in the Smoke Disk Basic. The Random Access of Percom SUPER BASIC is demonstrably faster than Smoke disk basic, is easier to implement, and does not depend on those accursed overlays! It is truly Random Access.

d) I question the statement that SMUG has read an Apple diskette using a Smoke disk system. FLEX and CP/M I believe but not Apple. Apple does not supply an Address mark and header format compatible with the 1771 controller. On the other hand FLEX, Smoke, CP/M and Apple diskettes are quite readable with the Percom controller.

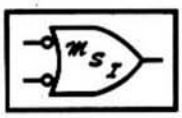
e) The sectors on the Percom disk are bidirectionally linked just like both Smoke and FLEX. (What's this REPAIR? A good DOS doesn't permit things like that to happen!)

Enough with the self defense. I had hoped my remarks in Issue #3 would be rebutted with more knowledge and objectively. There certainly is a case for the more sophisticated Disk Operating Systems particularly if one's task is system software development.

TABLE 1

	Simple System (PERCOM MINDOS-PLUSX™)	Complex System (SMOKE SIGNAL BFD-68)
Total Disk Storage Capacity (35-track Mini Disk)	89,600 bytes	80,640 bytes
Disk Capacity Used by DOS	512 bytes (max)	16,768 bytes (min)
Useful Disk Capacity	89,088 bytes	61,876 bytes
Named files?	Yes (31)	Yes
Random Access?	Yes	Yes
Single Drive Usefulness	Yes	Limited
Time to Save 10K Program	2 Seconds (max)	30 Seconds (min)
Time to Load 10K Program	2 Seconds (max)	10 seconds (min)
RAM required by DOS	600 bytes	6000 bytes
Cost of Assembled Single Drive Minidisk System Complete with Power Supply, Enclosure, named file DOS.	$599.95	$734.95
Cost of RAM to Support DOS (Prorated Smoke 16K RAM)	$ 12.34 (600 bytes)	$123.38 (6K bytes)
Cost of Random Access Disk BASIC	$ 49.95	$ 89.95
Total Cost	$662.24	$948.28

™MINIDOSPLUSX is a trademark of Percom Data Company, Inc.



Midwest Scientific Instruments, Inc.
220 West Cedar - OLATHE, KANSAS 66061
Phone 913 764-3273

June 5, 1979

Mr. Don Williams
"68" Micro Journal
3018 Hamill Road
PO Box 849
Hixson, Tennessee 37343

Dear Don,

I apologize for my delay in responding to your memo of 5/9/79, but it just now came to my attention.

Of course there are disadvantages of an operating system which lives in the middle of memory as does the MSI DOS. However, our FD-8 Disk Memory and associated MSI DOS software was the first operating system available for the 6800 Microprocessor. At the time it was introduced, memory was much more expensive than it is now and one of our design goals was to not require the user to have additional memory in his system . Most systems had 16K or so as a standard configuration at the time. Hindsight is a wonderful thing, and in retrospect I suppose it might have been better to locate the operating system high in memory so that it would not have been in anyone's way. However, we do make complete source code available for the operating system, and always have, and many of our users have relocated it to their satisfaction very easily.

I would refer you to a letter which was published in your journal Volume 1 issue 3 written by Mr. R. C. McKay of Inland Computers. There you will read another man's opinion of our FD-8 and Operating System.

Good luck with your new journal we hope to be doing some advertising with you ourselves soon. Thanking you very much for the opportunity to reply, I remain

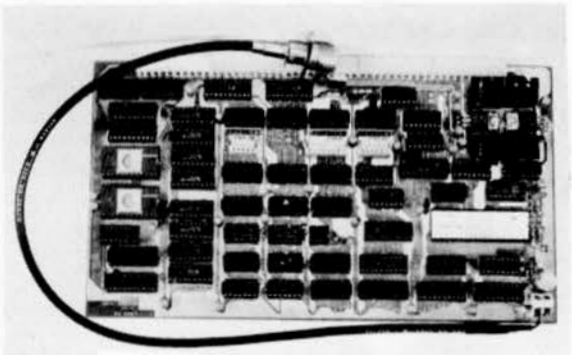
Sincerely yours,

Curt

Charles C. Childress, Ph.D.
President

CCC:rw

NEW PRODUCTS



SS 50 BUS 80 x 24 VIDEO BOARD

GIMIX INC., announces its versatile 80 x 24 video board with hardware scrolling, x-y addressable cursor and multiple character generators for the SS 50 bus that allows user-defined programmable character sets. It includes a TMS 2716 EPROM that contains a full 128 upper and lower case ASCII character set with true descenders; plus a socket for another TMS 2716 for an optional 128 character set; plus 2K of RAM for user-defined programmable character sets. This gives the user the ability to create his own hieroglyphics, alphabet, graphic elements, etc. and store them on PROM, disk, or tape.

The user can choose and intermix 384 different characters from any or all of the character generators and display up to 256 at one time, normally or inversely, and at full or half intensity, at any location on the screen. Contiguous 8 x 10 character cells permit solid lines and connecting patterns with user definable graphic elements.

It is addressable to any 2K boundary. GHOSTable addressing allows multiple boards at the same address, making it ideal for multi-user applications. Custom screen and character cell formats and European versions are available.

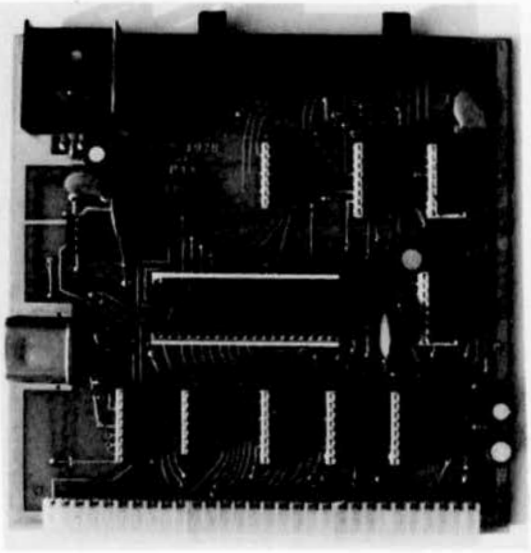
The available software includes a GMXBUG video based 3K ROM monitor, stand alone driver routines, and a program to create user defined characters.

Contact: Richard Don at GIMIX INC.
Phone (312) 827-5510
1337 W. 37th Place
Chicago, Illinois 60609

FLOPPY DISK CONTROLLER CARD ANNOUNCED BY GEORGIA COMPANY

An exciting new product, a floppy disk controller card, has been announced by Peripheral Technology of Columbus, Georgia.

The FD-1 controller card is designed for use in the S-50 buss with the controller designed for use in conjunction with the Shugart SA-400 minifloppy or its equivalent. The card can control up to three drives. Standard software, such as Flex 2.0



by Technical Systems Consultants, can be used without any patches.

Peripheral Technology offers not only a cost-effectivn price structure for this reliable product but also offers flexibility. The user may purchase his equipment either fully assembled or may elect to do the assembly individually. Price reductions are directly proportionate to the amount of assembly performed by the user. In addition, Peripheral Technology has recently reduced the price of the assembled board from $110 to $95. A completely assembled single drive system can be built for $475 or less.

A complete price and parts list is available upon request from: Peripheral Technology
3848 Hampton Drive
Columbus, Georgia 31904

PERCOM
PERCOM DATA COMPANY INC.

NEWS RELEASE

RELEASE: Immediate
CONTACT: Harold Mauch
(214) 272-3421

PERCOM UPGRADE CONVERSION Package TO RUN PERCOM SOFTWARE ON SOUTHWEST AND OTHER DISKETTE DISK SYSTEMS

Garland, Texas - June 6, 1979 - Harold Mauch, president of Percom Data Company, announced here today that the company has now available an upgrade/conversion package which allows owners of Southwest Technical Products Company and Smoke Signal Broadcasting Company mini-disk systems to use Percom disk software.

The conversion package includes Percom's LFD-400/800 Controller-Interface PC cards, two disk operating systems, FLEX DOS™ and MINI-DOS-FLEX™ on EPROM, Percom Super BASIC, a cable connector (for the SWTP disk cable), and, a full set of instruction and user manuals.

Mauch said the package is offered because many SWTP and Smoke users have been ordering Percom disk software.

He said Percom software for the 6800/6809 LFD mini-disk systems is more fully supported and easier to use than software offered by either SWTP or Smoke.

The conversion package sells for $249.95. Orders may be placed by dialing Percom's toll-free number: 1-800-527-1592. Payment may be made by check or money order or charged to Visa or Master Charge. Texas residents must add 5% sales tax.

Dealer inquiries are invited.

™ registered of Percom Data Company, Inc.

PERCOM DATA COMPANY 211 N. Kirby Garland, Texas 75042
(214) 272-3421

G2-BASIC

Dr. Chuck Adams
421 Frankie Ln.
Lewisville, TX 75067

REVIEW OF G2 BASIC WRITTEN BY MICROSOFT

GRT Corporation of Sunnyvale, Ca. (that's where the AMDAHL Corporation and many many other big electronic industries are in Silicon Culch) is now marketing Microsoft's BASIC under the trademark of G2 Standard Basic. This BASIC is an update of the earlier Microsoft 6800 Basic sold by MITS as Version 3.2. The updates have been of the nature to correct some minor problems and to add the cassette routines to save and load programs to cassette. The cassette and manual sell for $34.95.

There exists three basic classes of language processors for high level languages. These classes are

1. interpreters

2. compilers

and 3. incremental compilers.

Interpreters usually store the source statements in character form in memory and during the execution phase must scan each statement to determine the statement type and set up internal states to perform the desired operation or operations as described by the statement. Each statement must be rescanned each time it is executed, thus a great deal of overhead at execution time resulting in slow execution speeds for typical programs. One advantage to interpreters is that they typically run with small memory resources.

Compilers translate the source statements of a high level language such as BASIC into one or more machine language or assembly language instructions per source statement. If the compiler generates object code directly, then the object code may be saved to external devices such as tape or disk and later "loaded" into memory with additional routines, such as I/O routines, etc. and executed. Compilers are usually larger programs than interpreters and require additional resources such as disk to operate efficiently. The executable program is much faster and more compact and an additional level of security is obtained in that programs may be run in object form by others without the ability to recreate the original program, thus eliminating the disassembly reconstruction of code for the purpose of modification, etc.

Since the program in memory is encoded, Microsoft has chosen to dump the program to tape when performing a CSAVE function in binary format. Also the program is saved with a name preceding the program, the program name is given as part of the CSAVE command, i.e. CSAVE "pgm-name". The only disadvantage this author sees in this is that you should never forget the program name assigned, as there is no way to read that program back without the name, unless you go to extreme pains in reading and decoding the tape yourself. My suggestion is to name all the programs with your three initials, thus you never have to remember too much, except where on the tape the programs are recorded. Keep good records here.

The Basic has the commands as follows:

CLEAR	CSAVE "A"	LLIST (for printer)	PRINT
CLEAR n	CSAVE "A",S	NEW	RESTORE
CONT	INPUT	NULL	RUN
CLOAD "pgm"		RUN n	LIST

The manual is in error on the CLOAD, as it shows a CLOAD without "pgm-name" and an error is returned when this is tried using the BASIC.

Program statements in the basic are:

DEF DIM END FOR GOSUB GOTO LET ON IF POKE REM
RESTORE RETURN STEP STOP DATA READ

Functions available are:

ABS ATN COS EXP FRE INT LOG POS RND SGN SPC
SQR TAB TAN USR VAL PEEK ASC CHR$ LEFT$
LEN MID$ RIGHT$ STR$ WAIT

I think I got them all. It has everything that you need for writing of BASIC programs. The program takes up locations $0000 to $1FC2 when loaded into memory, and frees some of this up when execution of the BASIC is started. With 32K of memory, a message comes up that 25304 bytes of memory is available for programs and that should be plenty, since the programs will require less memory in the encoded form. I don't have a program that big and I wouldn't feel like keying it in anyway!

The only problem that the reviewer found is that the BASIC will not work with a SWTPCo system with a MP-C board for I/O. You will need some monitor other than MIKBUG and will have to have an MP-S board at Port 1 for proper operation of the software.

Shown below is the programs run and the times for same for the information of the reader. You may want to run the same programs on your system and basic for comparison.

Incremental-compilers lie midway between the interpreter and compiler in memory requirements and speed of execution of programs. The incremental-compiler takes each line of a program as it is typed into the system and translates it into a "pseudo-code", usually a one byte code identifying the type of statement and pointers to variable locations in memory or the variable names used for assignment or computation. At run-time a portion of the incremental-compiler handles the problem of execution by using the "pseudo-code", a one byte quantity, to perform the functions, thus eliminating the scanning of many characters in a line of source code and reducing the time required for the execution of a program.

The G2 BASIC V1.0 is an incremental-compiler, thus making it much faster than the existing basic interpreters on the market. Bill Gates and the group at Microsoft have used essentially the same design for previous implementations of BASIC on the *)*)(#@ systems. (Sorry but that was censored since this is a 6800 Journal). Benchmarks show that the 6800 version runs slightly faster than these other versions on the *)*)(#@ system and some are listed here for the readers consideration.

This basic uses a 6 digit floating point format, which may be disturbing to the business people, since they have to have 10 digits to balance their checkbooks, etc. Wish I had that problem.

The tape as it comes from G2 contains a loader program in MIKBUG format (MIKBUG is a trademark of Motorola Inc.) followed by the Basic in binary format. After loading into the system, the user may wish to dump the code entirely in ASCII format. I do include a patch at the end of this article for those who have the PERCOM disk which uses the low order 32 bytes of memory and so does the G2 BASIC. This is a holdover for the *)*) version I suspect, since all the true 6800 users use $0100 for the entry point, not $0000.

The BASIC is a good basic and should prove to be valuable to many of the readers who are not concerned with the available precision. It should provide faster response for many of the games written in BASIC and scientific type calculations involving single precision numbers of 6 decimal digits of accuracy. I don't know that G2 or Microsoft will come out with a version interfaced with a disk system for the 6800. This would require a significant cash outlay by someone and G2 is now swamped with Level III orders from many of the 200,000 owners of "the other system".

Good luck and may the Parser be with you.

```
A
 1          *
 2          *    PATCH TO G2 STANDARD BASIC
 3          *
 4          *    THIS PATCH ALLOWS STORAGE OF BASIC ON
 5          *    PERCOM DISK WITH ENTRY POINT AT $0100
 6          *
 7          *    WRITTEN BY CHUCK ADAMS
 8          *
 9          *    FEBRUARY 1979  (C) ALL RIGHTS RESERVED
10          *
11          *    THIS ROUTINE MAY BE USED FOR OTHER THAN
12          *    COMMERCIAL REASONS BY THE USER, AS OUTLINED
13          *    IN G2 COPYRIGHT
```

```
14                *
15         14E3 PRMPT  EQU   $14E3    START OF PROMPT MESSAGES IN B2 BASIC
16         0000 MEM00  EQU   $0000    START OF LOW MEMORY (CAN'T GET ANY LOWER)
17                *
18 0100                ORG   $0100    NEW ENTRY POINT
19 0100 7E 0020 START  JMP   BASIC1   GO TO INIT ROUTINE
20                *
21 0020                ORG   $0020    START OF SCRATCH AREA IN B2 BASIC
22 0020 CE 7E14 BASIC1 LDX   #$7E14
23 0023 DF 00          STX   MEM00    STORE IN LOW CORE
24 0025 CE E37E        LDX   #$E37E
25 0028 DF 02          STX   MEM00+2  INIT TWO BYTES AT A TIME
26 002A CE 1CFB        LDX   #$1CFB
27 002B DF 04          STX   MEM00+4
28 002F CE 7E03        LDX   #$7E03
29 0032 DF 06          STX   MEM00+6
30 0034 CE 5900        LDX   #$5900
31 0037 DF 0B          STX   MEM00+B
32 0039 4F             CLRA           GET READY TO CLEAR OTHER LOCATIONS
33 003A 97 0A          STAA  MEM00+10
34 003C 97 0B          STAA  MEM00+11
35 003E 97 0C          STAAMEM00+12
36 0040 97 0F          STAA  MEM00+15
37 0042 97 10          STAA  MEM00+16
38 0044 97 11          STAA  MEM00+17
39 0046 B6 4B          LDAA  $72
40 0048 97 0E          STAA  MEM00+14
41 004A 7E 14E3        JMP   PRMPT
42                     END

SYMBOL TABLE

PRMPT  14E3    MEM00  0000    START  0100    BASIC1 0020

   0 ERRORS
>
```

```
*Q
>BASIC


MEMORY SIZE?

TERMINAL WIDTH? 80
WANT SIN-COS-TAN-ATN? Y

25304 BYTES FREE

MICROSOFT 6800 BASIC
GRT VERSION 1.0
COPYRIGHT 1978 BY MICROSOFT

OK
CLOAD "BM1.0"

OK
LIST

100 REM
110 REM   BENCHMARK 1.0
120 REM
130 REM   TIME - 14.30 SECONDS
140 REM
150 FOR K=1 TO 10000
160 NEXT K
170 PRINT "END OF JOB"
180 END
OK



CLOAD "BM2.0"

OK
LIST

100 REM
110 REM   BENCHMARK 2.0
120 REM
130 REM   TIME - 181.0 SECONDS
140 REM
150 K=0
160 K=K+1
170 A=K/K*K+K-K
180 IF K<10000 THEN 160
190 PRINT "END OF JOB"
200 END
OK
CLOAD "BM3.0"

LIST

100 REM
110 REM   BENCHMARK 3.0
120 REM
130 REM   TIME - 32.75 SECONDS
140 REM
150 K=0
160 K=K+1
170 A=K/2.2*3.3+4.4-5.5
180 IF K<1000 THEN 160
190 PRINT "END OF JOB"
200 END
OK


CLOAD "BM4.0"

OK
LIST

100 REM
110 REM   BENCHMARK 4.0
120 REM
130 REM   TIME - 18.50 SECONDS
140 REM
150 K=0
160 K=K+1
170 A=K/2*3+4-5
180 IF K<1000 THEN 160
190 PRINT "END OF JOB"
200 END
OK

CLOAD "BM5.0"

OK
LIST

100 REM
110 REM   BENCHMARK 5.0
120 REM
130 REM   TIME - 14.3 SECONDS
140 REM
400 FOR K=1 TO 10000
```

```
500 NEXT K
700 PRINT "END OF JOB"
800 END
CLOAD "BM6.0"

OK
LIST

100 REM
110 REM   BENCHMARK 6.0
120 REM
130 REM   TIME - 98.5 SECONDS
140 REM
400 K=0
500 K=K+1
600 IF K<10000 THEN 500
700 PRINT "END OF JOB"
800 END
CLOAD "BM7.0"

OK
LIST

100 REM
110 REM   BENCHMARK 8.0
120 REM
130 REM   TIME - 2.70 SECONDS
140 REM
150 FOR K=1 TO 1000
160 REM FUNCTION TO BE EVALUATED
170 NEXT K
180 PRINT "*END OF JOB"
190 END
CLOAD "BM8.0"

OK

LIST

50 :REM   NOTE THE LOSS OF THE COMMENT 160 IS
100 REM                              IMPORTANT!!!
110 REM   BENCHMARK 9.0
120 REM
130 REM   TIME - 14.3 SECONDS
140 REM
150 FOR K=1 TO 10000
170 NEXT K
180 PRINT "*END OF JOB"
190 END
OK
```

CORRECTION!!

This is a correction for the same program EXPLOR; printed in the May 1979, Issue 3. Much as I HATE programs that are printed with an error in them; and all the yelling and screaming that goes along with that; I have produced a blooper of my own. I am very sorry for this And I can't think of good excuse for it, so I won't tell you one.

The error line reads: 1500 IF INT(D(V1,H1)<)3 THEN IF INT(D(V1,H1)) <)1 THEN 1370.

This should read; 1500 IF INT(D(V1,H1)))1 THEN 1370.

This will remove some strange responses when in the lower right quadrant of the rose maze. How about some cards and letters from those of you that have run that game; with some comments??

Respectfully
Tom Harmon
Box 493
Laurel, Md.
20810

SPOOL (SSB)

Dan Johnson
Solar Computer Corp.
7655 SW Cedarcrest St.
Portland, OR 97223

```
*     HERE'S ANOTHER INSTALLMENT IN MY CONTINUING EFFORTS TO EXTEND MY
SUBSCRIPTION.  THIS ONE IS A PATCH TO ALLOW YOU TO 'SPOOL' THE LISTING OUTPUT
FROM SSB'S SA-1 ASSEMBLER.  I.E. IT GIVES YOU THE OPTION OF SENDING THE
ASSEMBLY LISTING TO A DISK FILE INSTEAD OF THE PRINTER OR THE TERMINAL.
        TO USE: SAVE THE CODE SHOWN ON A DISK FILE.
                APPEND THIS FILE TO THE ASSEMBLER OBJECT CODE FILE.

THE 'P' SWITCH TO SEND THE LISTING TO THE PRINTER STILL WORKS AS USUAL;
TO SEND THE LISTING TO A FILE REPLACE THE 'P' WITH A <FILE-SPEC>.
```

```
MAL/6800 1.2: 0000   ASMPAT                    PATCH TO SSB SA-1 ASSEMBLER
17-MAY-79 13:41:59: Page 1: Pass 1

                      2:          NAM     ASMPAT
                      3:          WITH    WI=80
                      4:
                      5: ********************************************************
                      6: *Patch for SSB SA-1 ASSEMBLER to allow option of
                      7: *sending the assembly listing to a disk file;
                      8: *instead of the line-printer.
                      9: *by:    Dan Johnson
                     10: *       Solar Computer Systems Corp.
                     11: *       7655 S.W. Cedarcrest St.
                     12: *       Portland, Or.  97223
                     13: ********************************************************
                     14:
0030                 15: XTEMP6  EQU     $30
72FC                 16: IBFPNT  EQU     $72FC
161D                 17: ZANCHK  EQU     $161D
1620                 18: ZFLSPC  EQU     $1620
1629                 19: DFM     EQU     $1629
0002                 20: OSWRIT  EQU     2
0001                 21: OD4W    EQU     1
0001                 22: FTCB    EQU     1
0000                 23: XFC     EQU     0
000C                 24: XFT     EQU     12
1626                 25: ZTYPDE  EQU     $1626
162C                 26: CDFM    EQU     $162C
1623                 27: ZWARMS  EQU     $1623
166F                 28: DABNB1  EQU     $166F
                     29:
                     30: ********************************************************
                     31: *WITH PATCH INSTALLED ASMB MAY BE CALLED AS FOLLOWS
                     32: *ASMB,<SOURCE-FILE>,<OBJECT-FILE>,<LIST-FILE>,<N>,<N>
                     33: ********************************************************
                     34:
                     35:          ORG     $163C
163C 18A2            36:          FDB     BUFBEG             NEW BUFFER BEGINNING
                     37:
                     38:          ORG     $16B2
16B2 7E1AA7          39:          JMP     APAT1              JUMP TO PATCH
                     40:
                     41:          ORG     $1AA7
                     42:
1AA7 FE72FC          43: APAT1   LDX     IBFPNT             GET LINE PNTR
1AAA 08              44:         INX
1AAB A600            45:         LDA A   0,X
1AAD BD161D          46:         JSR     ZANCHK             SEE IF ALPHA-NUMERIC
1AB0 2403            47:         BCC     DETNAM             IF SO
1AB2 7E166F          48:         JMP     $166F
1AB5 CE1AFC          49: DETNAM  LDX     #LFCB
1AB8 BD1620          50:         JSR     ZFLSPC             GET FILE NAME OF LIST FILE
1ABB 2529            51:         BCS     NOZAIN             IF ERROR
                     52:         LDX     LFCB+3             1ST 2 CHAR. OF NAME
1ABD FE1AFF          53:         CPX     #'P*256+256        'P<NULL>'?
1AC0 8C5000          54:         BNE     NOLINE             IF NOT PRINTER
1AC3 2605            55:
1AC5 8650            56: DETOUT  LD  A   #'P
1AC7 7E16B6          57:         JMP     $16 6              HE DOESN'T WANT TO SPOOL!
```

```
MAL/6800 1.2: 1ACD   ASMPAT                    PATCH TO SSB SA-1 ASSEMBLER
17-MAY-79 13:41:59: Page 2: Pass 1

1ACA CE1AEF          57: NOLINE  LDX     #SPOOL
1ACD FF1633          58:         STX     $1633              LISTING VECTOR
1AD0 CE1AFC          59:         LDX     #LFCB
1AD3 8601            60:         LDA A   #OD4W
1AD5 A700            61:         STA A   XFC,X              SET FCB FOR OPEN
1AD7 8601            62:         LDA A   #FTCB              SET FOR COMPRESSED TYPE
1AD9 A70C            63:         STA A   XFT,X
1ADB BD1629          64:         JSR     DFM                OPEN LIST FILE
1ADE 2506            65:         BCS     NOZAIN             IF ERROR
1AE0 8602            66:         LDA A   #OSWRIT
1AE2 A700            67:         STA A   XFC,X              SET FOR WRITE
1AE4 200F            68:         BRA     DETOUT             EXIT PATCH
                     69:
                     70: *ERROR ROUTINE
1AE6 BD1624          71: NOZAIN  JSR     ZTYPDE             TYPE ERROR MESSAGE
1AE9 BD162C          72:         JSR     CDFM               CLOSE ALL FILES
1AEC 7E1623          73:         JMP     Z'AMB              EXIT TO OP SYSTEM
                     74:
                     75: *ROUTINE TO SEND BYTES MEANT FOR THE LINE PRINTER
                     76: *TO THE LISTING DISK FILE
1AEF DF3C            77: SPOOL   STX     XTEMP6             AVE INDEX REG
1AF1 CE1AFC          78:         LDX     #LFCB
1AF4 BD1629          79:         JSR     DFM                SEND BYTE TO FILE
1AF7 26ED            80:         BNE     NOZAIN             IF ERROR
1AF9 DE30            81:         LDX     XTEMP6             RESTORE INDEX
1AFB 39              82:         RTS
                     83:
1AFC 00A6            84: LFCB    RMB     166                FILE BUFFER
                     85:
18A2                 86: BUFBEG  EQU     *
                     87:         END
```

Symbols Sorted by NAME:

APAT1/1AA7	BUFBEG/18A2	CDFM/162C	DABNB1/166F	DFM/1629
FTCB/0001	DETNAM/1AB5	DETOUT/1AC5	IBFPNT/72FC	LFCB/1AFC
NOZAIN/1AE6	NOLINE/1ACA	OD4W/0001	OSWRIT/0002	SPOOL/1AEF
XFC/0000	XFT/000C	XTEMP6/0030	ZANCHK/161D	ZFLSPC/1620
ZTYPDE/1626	ZWARMS/1623			

Symbols Sorted by Value:

XFC/0000	FTCB/0001	OD4W/0001	OSWRIT/0002	XFT/ 00C
XTEMP6/0030	ZANCHK/161D	ZFLSPC/1620	ZWARMS/1623	ZTYPDE/1626
DFM/1629	CDFM/162C	DABNB1/166F	APAT1/1AA7	DETNAM/1AB5
DETOUT/1AC5	NOLINE/1ACA	NOZAIN/1AE6	SPOOL/1AEF	LFCB/1AFC
BUFBEG/18A2	IBFPNT/72FC			

22 Symbols.

XBAK-CMD

ANYONE DOING A LOT OF TEXT EDITING WITH TSC'S 'FLEX' DISK EDITOR PROGRAM CAN APPRECIATE THE VALUE OF THE THE EDITOR'S FILE BACKUP FEATURE. HAVING A BACKUP FILE HAS RESCUED ME MORE THAN ONCE FROM AN OVERLY AMBITIOUS EDITING ADVENTURE.

AFTER AN EXTENDED EDITING SESSION, HOWEVER, YOU MAY FIND YOUR DISK CLUTTERED WITH A NUMBER OF UNNEEDED BACKUP FILES. THESE M UST BE DELETED INDIVIDUALLY WITH THE ' KLETE' COMMAND.

I'VE ADDED AN ADDITIONAL COMMAND TO MY SYSTEM CALLED 'XBAK' WHICH WILL AUTOMATICALLY DELETE ALL FILES WITH THE EXTENSION OF 'BAK'. THIS TIME-SAVING PROGRAM TURNED OUT TO BE VERY SIMPLE TO IMPLEMENT.

THE 8" 'FLEX' DOS (I DON'T KNOW ABOUT THE 'MINIFLEX') ALREADY HAS THE COMMAND 'XOUT' WHICH DELETES ALL FILES WITH THE EXTENSION OF 'OUT'. THE PROCEDURE FOR CONVERTING 'XOUT.CMD' TO 'XBAK.CMD' IS A SIMPLE ONE. ALL THAT IS INVOLVED IS FINDING THE PLACE WHERE 'XOUT' LOOKS FOR THE CHARACTERS 'O','U' AND 'T' AND CHANGING THESE TO 'B','A' AND 'K'. THE FOLLOWING STEPS WILL DO JUST THAT:

FIRST, LOAD 'XOUT.CMD' INTO MEMORY AND RETURN TO THE MONITOR)

```
         ***GET XOUT.CMD
         ***MON
```

THEN CHANGE $A158 FROM $4F TO $42
$A161 FROM $55 TO $41
$A167 FROM $54 TO $42

NOW SAVE THE MODIFIED COMMAND ON DISK:

```
         ***SAVE.LOW XBAK.CMD A100 A1FA A100
```

AND YOU'RE DONE! NOW WHAT COULD BE EASIER? A NEW UTILITY COMMAND AT NO COST, NO EFFORT, AND IN PRACTICALLY NO TIME AT ALL.

TO USE IT, JUST TYPE 'XBAK' AND ALL BACKUP FILES WILL BE DELETED. (JUST BE SURE TO RENAME ANY BACKUP FILES THAT YOU MAY WANT TO KEEP.)

John K. Jordon
103 Elliott Cir.
Oak Ridge, TN 37830

The following is a little puzzle that runs under BASIC, and was written by Gary Holowicki.

The words are hidden, and you have to find them. They may be found by reading... left to right, right to left, top to bottom, bottom to top, on the diagonals, both directions.

Have fun.....

Reprint from '6800 BITS'
Chicago Area 6800 Newsletter
Phil Schuman, Editor

```
T R 6 8 0 9 E Y M U F J D D R
N O P V J X G F R W D A X P M
L F L X X V U P N Z V S N V T
A M V Z Y N B U P S O G K Y C
N K S 9 C Q K F B N M B Y X M
G Z B J O T I L E W T O J M S
I V T S Z C M E C S P C K H R
S W E S Z Z I X C E X P Q E D
X Y P W J X Q S R S L R A Z X
B A E T T U N C A W C G K S N
W D Q P H B O O N B D E B Z A
A K E C V M O W E N V R F F B
D M P D Q 8 B P P D R K T N I
V R Q Y 6 I C J O P Z X T A S
E S T I T E L J O T D M R T S
```

THE HIDDEN WORDS ARE:

6800	6809
TSC	SWTPC
SMOKE	SIGNAL
PERCOM	BASIC09
FLEX	MIKBUG

JUMBLE-(Basic)

This BASIC program generates and prints all permutations of strings entered to it. This assists the user to solving the JUMBLE puzzle in the newspaper. It is based upon Algorithm 115 (PERM) originally published in the Communications of the ACM in August 1962 on page 434 and written by R. F. Trotter. The BASIC program was coded for a SWTPC 6800 in 8K BASIC; however, it should run on any BASIC which supports strings. Since the number of permutations of a string of length N is N! (N-factorial), the following table provides the number of permutations and minimum time to print all permutations of a string of length N (assuming negligible processing time and 30 CPS terminal):

N	N!	Time
3	6	1 second
4	24	5 seconds
5	120	26 seconds
6	720	192 seconds (3 minutes, 12 seconds)
7	5040	1512 seconds (25 minutes, 12 seconds)
8	40320	13440 seconds (3 Hours, 44 minutes)

It is obvious that this approach is not practical for strings of length greater than six. Luckily, most of the strings in the JUMBLE puzzle are of length five or six.

Several extensions of this algorithm are possible. One is to compare each newly-generated word with all previously-printed ones to minimize the number of words actually printed in case of redundant letters. This would probably work well on a large time-sharing system; however, the time and space requirements on a micro-computer system render this approach infeasible. This concept is good, and it is conceivable that an algorithm could be devised which would generate only the non-duplicated permutations without an exhaustive search of previously-generated permutations. Another idea is to add a kill-list of forbidden consonant pairs such that if a word contained one of the forbidden pairs (in adjacent positions), it would not be printed. This would eliminate the printing of many of the words, would not be prohibitive in space, and may not be prohibitive in time in some implementations of BASIC, though it probably would be in SWTPC 8K BASIC. Still another idea would be to allow the specification of certain characters as being required to be in certain positions in acceptable words, though which characters were in which positions would still remain unspecified. This is the most specific extension and has the best apparent chance for major improvement in a BASIC implementation which is rather time-efficient. Either of the latter suggestions would probably materially improve the algorithm if it were rewritten into assembly language or run using BASIC compilers.

The next time you work a JUMBLE puzzle, use the following algorithm to help you try to generate the words.

Dr. E. M. Pass
1454 Latte Lane
Conyers, GA 30207

```
'0.1ST
0100 DIM P(10),D(10),X$(10)
0200 PRINT "ENTER NUMBER OF CHARACTERS";
0300 LET F=1
0400 INPUT M
0500 PRINT "ENTER STRING, ONE CHARACTER AT A TIME"
0600 FOR I=1 TO M
0700 INPUT X$(I)
0800 LET P(I)=0
0900 LET D(I)=1
1000 NEXT I
1100 LET K=0
1200 LET N=M
1300 LET Q=P(N)+D(N)
1400 LET P(N)=Q
1500 IF Q<>N THEN 1600
1600 LET D(N)=-1
1700 GOTO 2100
1800 IF Q<>0 THEN 2600
1900 LET D(N)=1
2000 LET K=K+1
2100 IF N<3 GOTO 2400
2200 LET N=N-1
2300 GOTO 1300
2400 LET Q=1
2500 LET F=0
2600 LET Q=Q+K
2700 LET T$=X$(Q)
2800 LET X$(Q)=X$(Q+1)
2900 LET X$(Q+1)=T$
3000 FOR I=1 TO M
3100 PRINT X$(I);
3200 NEXT I
3300 PRINT " ";
3400 IF F=1 THEN 1100
3500 PRINT
3600 END
```

```
KSI READY
#RUN
ENTER NUMBER OF CHARACTERS? 4
ENTER STRING, ONE CHARACTER AT A TIME
? A
? B
? C
? D
BACD  BCAD  BCDA  CBDA  CBAD  CABD  ACBD  ACDB
CADB  CDAB  CDBA  DCBA  DCAB  DACB  ADCB  ADBC
DABC  DBAC  DBCA  BDCA  BDAC  BADC  ABDC  ABCD
```

SD-BASIC COMPILER

William E. Fisher
132 Brookmeade Dr.
Pittsburg, PA 15237

Convinced of the revolution microprocessors would make upon the business community, the author purchased a SWTP 6800 cassette system in the fall of 1976. This 16K system was upgraded with the SWTP FDOS mini-disks in September, 1977, later enhanced by Mini-Flex, and finally replaced by a 40K SWTP DMAF-1 disk system running under Flex in December, 1978. We are currently supporting several local businesses by writing new applications, installing new purchased software packages or I/O devices, and providing ongoing maintenance for production programs.

After writing several accounting packages in various configurations of Basic, several problems or bottlenecks became apparent.

Complex programs that required a large number of variables became "unreadable" shortly after having been installed. This was due to the fact that most programmers find it difficult to "read" programs with typically meaningless variables such as B, Z9, Q$, etc. This becomes a real problem when customers want fast turnaround on modifications to their systems since the programmer must again become reacquainted with a rather meaningless bunch of gibberish before a change can be attempted.

Even a simple program deserved good documentation. This meant plenty of REM statements. In a business environment, with many programs averaging 125-150 lines, the necessary REM statements could increase this to 200-250 lines of coding. Unfortunately REM statements wasted valuable memory forcing the customer to have more memory than actually needed to do the job. In addition, REM statements in an interpretive Basic degraded system performance.

The lack of PRINT USING and FORMAT commands was a definite handicap. To compensate for same, complex subroutines had to be copied into the program.

Restrictions imposed by LINE NUMBERS consistently contributed to the overall decrease in programmer productivity.

There was little protection of our investment since the customer could easily convert the software to another system or could modify the software themselves.

With the above points in mind we began searching for a more powerful tool for our business applications. After considerable investigation and aborted attempts with someother products, we are now using the SOFTWARE DYNAMICS BASIC COMPILER. This product offers features and performance unmatched by software offered by other 6800 vendors. Some of it's highlights are as follows:

The Basic program is first compiled by the SOFTWARE DYNAMICS BASIC COMPILER. The output from this step is then assembled by their MAL ASSEMBLER. The final result is an abbreviated code much like the pcode used with Pascal. Although a runtime package is utilized, the overall utilization of memory is much less than with an interpretive Basic. Even with the runtime package, one can usually execute a 225 line program in 16K and 1100 lines of code in 32K.

Remarks statements are not compiled, again a memory saving. And since they are dropped in the compilation process, they do not degrade the system's performance. In fact, the often quoted Kilobaud benchmark #7 runs in 26.5 seconds with the older 898,550 Hz clock in the original SWTP 6800 systems.

Another nice feature is that any indenting or margins used by the programmer are

maintained in the output listing. Very nice for structured programs and IF THEN ELSE conditionals.

Variable names can be up to 15 characters in length. A typical Basic statement of LET N = O + D - C can now be coded into a more meaningful LET NEWBALANCE = OLDBALANCE + DEPOSITS - CHECKS.

Line numbers are not required except as the targets for GOTO's and GOSUBS's. Program can be more easily modified when the programmer doesn't have to be concerned with trying to "shoehorn" in a few statements between line numbers. The retyping of complex statements just to change some line numbers has been eliminated. In addition, the line numbers do not have to be in ascending sequence. SOFTWARE DYNAMICS has a new version 1.4 (which may already be available when you read this) which does not require line numbers. Instead you use names like CHKBALANCE4ZERO, TEST4OVERDRAWN, or GIVECASHREFUND.

PRINT USING & FORMAT statements - these powerful commands make the formatting of complex print statements a simple task. Subroutines to convert, right justify, and float dollar and credit signs for monetary amounts have been eliminated. For instance:

```
The statement PRINT USING 200, 1.27, 1.27, -1.27, -1.27
formatted by 200 FORMAT "$##.## : $-##.## : -$##.## : $##.##-"
would print $1.27 : $1.27 : $-1.27 : $1.27-.
```

The IF THEN ELSE structure is provided, can use multiple lines of coding, and can be compounded. FOR NEXT loops can nest as deeply as desired. This is controlled by the I/O package which interfaces the software to FLEX and can be easily changed if desired. The source code for the I/O package is provided on a diskette and the manuals explain any steps necessary to add new devices on other ports, increase the size of the FOR/NEXT loop stack, etc.

Unlimited string lengths are another convenience. The only restriction on string lengths is the memory size of the user's system. Singly and doubly dimensioned arrays are supported, again limited only by memory size.

The usual string processing commands are available. In addition, the FIND command permits rapid and simple substring searches. When editing input transactions this feature is extremely valuable, i.e.

```
If the string INPUTRANCODE$ = "add"
and the string VALIDTRANCODES$ = "ADDaddCNGcngDELdel"
then execution of X = FIND(VALIDTRANCODES$,INPUTRANCODE$)
```

would result in X containing the value 4 since the input transaction code was found in the string starting with the 4th character. If the tran code was not found, then X would equal 0. Suppose you need to determine if an input string contains a comma. The statement X = FIND(INPUTSTRING$,",") would return a result of 0 if the comma is not present, and if it is, the position first encountered in the string INPUTSTRING$. Using the ON GOTO or ON GOSUB commands in connection with FIND permits rapid vectoring to subroutines based upon tests of input data.

The compiled program cannot be easily modified by the end user. This provides relief from selfstyled "helpers" at the customer's location and insures additional revenues for future program enhancements or maintenance. All object (machine) code produced is ROMable. For the more advanced programmer, the RUNTIME package is reentrant and tolerates interupts.

Another powerful feature is the ability to specify the origin of your program and/or it's data. This allows a program to process data and values prepared by a preceeding program and to pass results to a succeeding program.

The ON ERROR command gives the programmer the ability to trap errors occurring during

execution and to programmatically recover. This can minimize or eliminate the chance of the end user having a useless error # and READY prompt appearing on the console. After all, the end user would probably not know how to recover from such a situation and it should have been handled by the programmer. And in a compiled program environment, it is usually mandatory that all possible errors be anticipated and handled by the program.

The ability to AND, OR, XOR, and shift on a bit/byte level will be of use in certain applications. In addition, the ability to specify hexadecimal constants and being able to PEEK and POKE hex addresses is a definite improvement.

The package has been fully interfaced to the FLEX operating system. File commands include OPEN, CLOSE, CREATE, DELETE, RENAME, RASCII, RBIN, WASCII, WBIN, POSITION, and CHAIN. Reading and writing of ASCII or binary data is certainly a convenience. And the ability to position the file pointer to ANY byte within ANY record is a must in processing random records.

The SOFTWARE DYNAMICS products are also available for the American Microsystems, Electronic Product Associates, Motorola Exordisk, Midwest Scientific Instruments, Smoke Signal Broadcasting, and Wavemate systems.

One drawback is the necessity to recompile the program whenever a change is desired. The ability to make a fast fix and RUN is not available. However, careful planning of the change will allow its implemention in an orderly manner which will not detract from the original program's clarity, and should yield dividends in staff productivity when enhancements are desired.

I have found the Software Dynamics staff to be very helpful and quite responsive to questions, problems, and enhancements. Their literature states "We're here to help" and I have always found this to be an understatement of their response to my inquiries. In addition, I have found the software has provided increased programmer productivity and a drastic reduction in run times for many non I/O bound programs doing string manipulations, array processing, and limited number crunching.

A man once said "Time is money!". Wouldn't it be nice to eliminate many of the aggravations imposed upon you since you were forced to join the "Basic" crowd? Lastly, can you afford not having this degree of performance?

The SOFTWARE DYNAMICS BASIC COMPILER, MAL ASSEMBLER, I/O PACKAGE and over 200 pages of documentation and examples retails for $350. Interested parties should contact Software Dynamics, 2111 W. Crescent St., Suite G, Anaheim, Ca. 92801. Phone (714)635-4760.

DATA BASE/SORT-MERGE

Paul Searby
1512 Encinitas Blvd.
Encinitas, CA 92024

The purpose of this article is to familarize the reader with some of the Data Base and Sort/Merge concepts that are being used in many of the application software packages available on the 6800. Parts of the article are excerpts from a manual I just wrote for a generalized Random Access Data Base Sort/Merge software package which was written with Computerware* Random Access Basic. Many of you might ask: Why is a sort or merge program so important? The answer to this partly depends upon what you are trying to accomplish. If it does not involve data files with a potentially large number of entries and several pieces of information for each entry - then the answer may be: They are not! If it does involve files like this, then invariably the user of this file is going to want to see the information in the file

ordered in more than just one way. For example, take a name and address file that also has date information included. He probably will want to list the people in the file in alphabetical order, Zip code order and by the date field. Do we keep three files - how do we insert new entries - or do we use a Sort/Merge to arrange the people in the right order when we print the lists? This is just one of many examples of the usefulness of a Sort/Merge that interfaces with all our Basic programs perfectly because it is written in Basic. So much for a reason - let's look at some of the concepts involved.

Random Access Basic's purpose in life is to allow the user to be able to create Data Bases in which all the information about a particular 'thing' can be stored and accessed/updated quickly. Just what is a Data Base?? For our purpose, it's a logical grouping of information about some 'thing'. It can be accessed rapidly and can either be read or written into directly with out having to make a second file. What is a 'thing'?? It's a subject or area of interest relevant enough to warrant having a Data Base. (Inventory, People, Wine). The information inside a Data Base can be accessed many ways: 1) Directly by Record Number (RECNO); 2) Sequentially; 3) By the use of a hashing algolrithm; 4) By means of a KEY File - plus many more.

If you assign the RECNO of a record (may contain many variables) as its 'Item number' and you always print the item number along with the record's real name, you can teach the user to use 'item number' when referencing that record and thus be able to go directly to the information that is wanted. The result is instantaneous information retrieval, which is important if a lot of requests or 'updates' are being made. Also, the Data Base does NOT have to be maintained in any particular order - new entries are added in the first available slot, allowing for efficient space usage.

Sequential access to a data base can be used when accumulating summary information, mass changes or extracting data for Key files. Our applications typically use sequential access only for extracting data, and to date, we have not found a need for hashing techniques in our use of random access files.

A Key File is typically a sequential file with one numeric and one string variable. The numeric variable contains the Record Number (RECNO) of one of the 'master file' records, and the string contains one or more variables concatenated together for the purpose of ordering (sorting) the records in the 'master file'. Since the 'master record' may contain 10 to 20 variables, it is much more efficient to sort only the record number rather than the whole record. With Random Access, we'll have no problem getting the whole record at print time, when the information is needed. We can make our sorting job even easier by exercising any selection processes prior to the sort and eliminating the records not wanted from the Key File.

The previous discussion implies that a critical part of the whole Data Base - Sort/Merge process is the Extract Program - at least I hope so, because it is. The Extract Program is a multi-dimension traffic cop in that it directs which, if more than one, print or other type of program will run after the Sort/Merge --- and which records in the Data Base will be extracted for the Sort/Merge. A good example of Program direction can be found in the CSS RANDOM V-2 MAILING SYSTEM and the Sample Extract progam in the SORT/MERGE or MAILING SYSTEM has good examples of record selection.

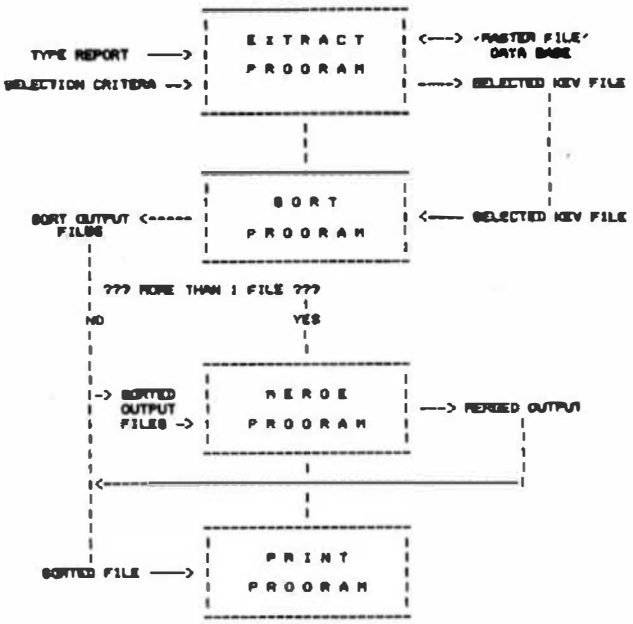
To summarize the process, first we extract the record number and

sort key of the records we want to report or whatever - then we enter the generalized Sort/Merge system to order the records as we want them - then we run the report or whatever, using the RECNO's in the sorted extract file to locate our 'master file' records in the desired order.

As a final note, I will try to answer a couple of the often asked questions concerning Data Bases and Sort/Merges. 1). Isn't a sort written in Basic too slow? - To answer this you must look at the total Extract/Sort/Merge process - in fact the actual Sort portion of the Whole is around 20%. Yes, a Sort written in Basic is slower than one written in machine language, but even a 100% improvement in the sort is only about a 10% improvement in the total run time. The advantages of easy interface and ease of modification that come with a Sort written in Basic typically outweighs the minor speed advantage gained from a machine language sort. The above has been empirically proven, by the way. 2). Isn't it inefficient to always sort instead of keeping the data in correct order? - Again, the total picture must be examined - what is 'correct' order? If the data will never be used in more than one order, then yes, sorting each time is inefficient. My experiences lead me to believe that most people want to see information formatted more than only one way - to get a different perspective. Thus flexibility and ease of modification become as much or more important than just speed. If you implement date last changed logic into the Data Bases, then the prior sorted key file may be re-used and the sort eliminated.

The above is one of several short articles about Data Base Concepts that I plan on writing in the next few months. Next, will be on Soft 'Record' Chained Random Data Bases that give you the ability to have Variable extension records for a Master Data Base and automatic maintence of the Chained Data Base while deletions, additions and changes are taking place. If I don't write it down soon, nobody will remember how it works!

SORT/MERGE SYSTEM FLOW



6800-CLEAR MEMORY

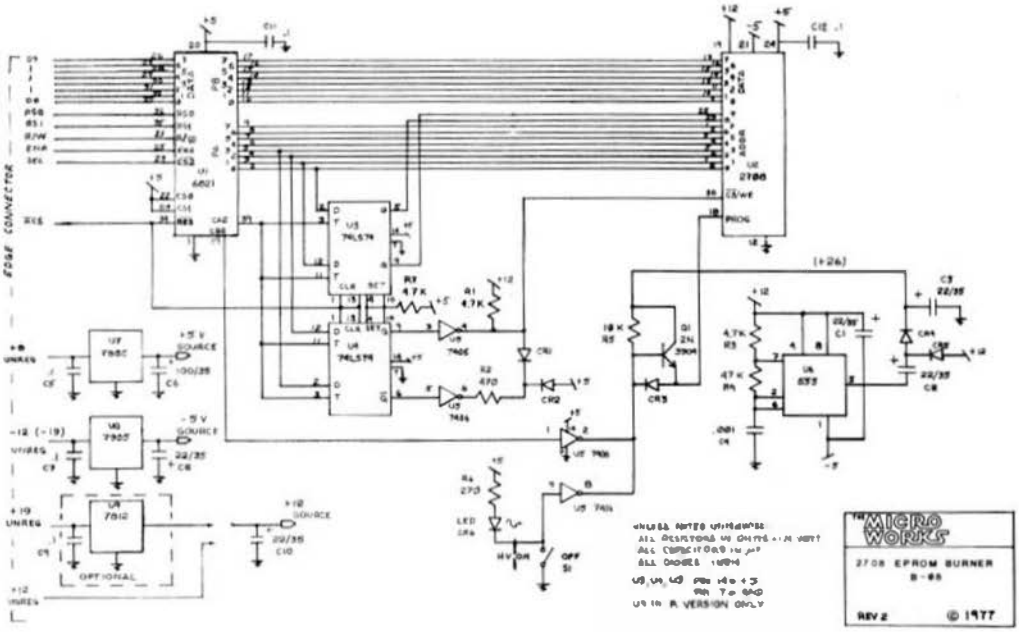
Brian Abernathy
729 Shannon Dr.
Marietta GA 30066

This is a simple routine that I developed late one evening after having several experiments run away with me, rather than let them run rampant through memory, and destroy the programs that I had already developed, I preface all my experiments with this short program which places a '3F' in every memory location.

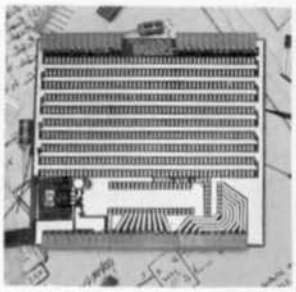
```
0000 86 3F    LDAA #3F          PLACE '3F' IN ACCUMULATOR A
0002 8E FFFF  LDS  #FFFF        SET STACK POINTER TO TOP OF MEMORY
0005 36       PSHA              PUSH A
0006 20 FD    BRA  #FD          BRANCH BACK ONE BYTE
```

Notice that when the program comes to address 0007 (the second byte of the BRA instruction) the next address that it will jump to will be 003F - and cause an interrupt - and jump back to the system monitor. If you feel that the system must be "clean", you can go into memory and clear the other 7 bytes manually. This has saved a long program from becoming "garbage" more times than one!

Build Your Own 2708 Programmer!!
HERE'S HOW:



THE MICRO WORKS P.O. BOX 1110, DEL MAR, CA 92014 (714) 756-2687



And here's the wire wrap board to build it on. The Universal I/O Board has space for a 40-pin wire wrap socket for any of Motorola's 40 or 24 pin interface chips; data and control lines are connected to appropriate edge connector pins. All other bus connections are brought out to a 16 pin socket pad. UIO: $24.95

Software? A stripped down version of our U2708 Utility program was published in *Dr. Dobb's Journal of Computer Calisthenics & Orthodontia*, February issue, #32. Or the complete version of U2708 is available in the B-08 Owner's Manual for only $10.00.

BUT...

If you're not a dyed in the wool wirewrapper and would like to purchase

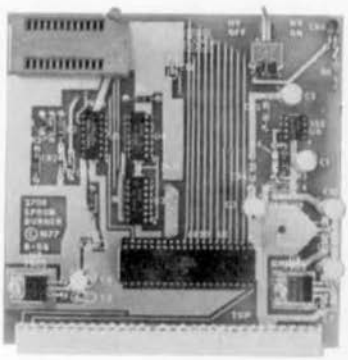
The Real Thing

Assembled, Tested, Burned in and

★ READY TO GO ★

GOTCHA COVERED

B-08 Eprom Programmer $99.95
U2708 Software In Eprom $29.95



COMPUTERWARE

HAS HARDWARE!

ITEM	PRODUCT DESCRIPTION	PRICE
CHIEFTAIN I	- SSB COMPUTER - 32K MEM DUAL MINI-FLOPPY - TWO SERIAL I/O 2 MHZ CLOCK - 9 SLOT MOTH R BOARD	$2595.00
CHIEFTAIN I V	- SAME W/DUAL SIDED 5" DISKS	$2950.00
CHIEFTAIN II	- SAME W/DUAL 8" DI KS	$3495.00
MPU-68	6800 W/32K, SMARTBUG, PS-I & CA LE	$1325.00
MP-6 /2	SWTPC COMPUTER SYSTEM W/8K MEM	$ 595.00
MP-8M	SWTPC 8K MEMORY	$ 250.00
M-16A	SSB 16K MEMORY	$ 299.00
MP-LA	SWTPC PARALLEL I/O	$ 49.00
MP-S	SWTPC SERIAL I/O	$ 49.00
PAR-I	SSB PARALLEL I/O	$ 39.00
ER-2	SSB DUAL SERIAL I/O (HANDSHAKE)	$ 89.00
MP-R	SWTPC 2716 PROM PGR	$ 60.00
MUSIC	NEWTECH MUSIC BOARD	$ 59.95
8-	MW 2708 PROM PGR	$ 104.95
PS8-08	MW 2708 PROM BOARD	$ 124.95
DS-68	MW DIGISECTOR	$ 179.95
UIO	MW UNIVERSAL I/O BOARD	$ 24.95
IO-120	SUROC 24 X 80	$ 995.00
ACT-IA	MICR -TERM 16 X 64 UC & LC	$ 400.00
VM09	SANYO 9" MONITOR	$ 185.00
VM12	VIDEO 100 12" MONITOR	$ 149.00
ACI-33	PCC CASSETTE I/O	$ 59.95
ST-45	SANKYO CA SETTE RECORDER	$ 39.95
BFD-68	SMOKE SIGNAL BROADCASTING FLOPPY DISK SYSTEM 1 - 3 DRIVES	$ 795.00 $1139.00 $1479.00
LFD-1	SINGLE LARGE FLOPPY DISK SYSTEM	$1395.00
LFD-2	DUAL LARGE FLOPPY DISK SYSTEM	$1895.00
BFD-2	DUAL SIDED LARGE FLOPPY DISKS	$2495.00
ABFD-68	BFD-68-1 WITHOUT CASE & PS	$ 649.95
MF-68-U	SSB UPORADE FOR SWTPC DISK	$ 269.00
PR-40	SWTPC PTR A SEM W/CVR & CABLE	$ 400.00
RIB ON	RIBBON FOR PR-40	$ 3.00
BP- 000	ANADEX 112 CPS 80 COL BI-DIRECTIONAL SPROCKET FEED	$ 995.00
DP-8000/R	ANADEX W/3K OPTIONAL BUFFER	$1045.00
702	CENTRONICS 120 CPS 132 COL BI-DIRECTIONAL - TRACTOR INCL 9 X 9 MATRIX UPPER & LOWER CASE	$2200.00
703	CENTRONICS 180 CPS 132 COL BI-DIRECTIONAL - TRACTOR INCL 9 X 9 MATRIX UPPER & LOWER CASE	$2800.00
RC-68	6800 CPU ROM/PROM CONVERSION 5.25" DISKETTES 10/BOX 8.00" DISKETTES 10/BOX CENTRONICS PRINTER CABLE	$ 17.95 $ 45.00 $ 55.00 $ 35.00

ALL PRODUCTS LISTED ARE NORMALLY STOCKED ITEMS.
OTHER ITEMS IN STOCK - CALL OR WRITE FOR INFO

THESE ARE CASH PRICES - ON PRE-PAID ORDERS OVER $25 WE WILL PAY HIPPING - BANK CARD, COD'S OR PURCHASE ORDERS. SHIPPING AND HANDLING EXTRA

COMPUTERWARE
(714)436-3512

1512 Encinitas Blvd.
Encinitas, CA. 92024

ED SMITH'S SOFTWARE WORKS

ANNOUNCES RRMAC

AN M6800 SYSTEM RELOCATABLE RECURSIVE MACROASSEMBLER & LINKING LOADER

ATTENTION ALL PROGRAMMERS: If you have been looking for an assembler with real macro capabilities, then RRMAC is the one for you.

Character replacement is the crux of macro expansion. RRMAC allows unlimited character replacement in any field of a macro model statement. RRMAC's argum nt notation can specify replacement from any field, subfield, sublist or substring of the macro call statement. RRMAC's set of macro directions supports both global and local set symbols. Set symbols can be based on either arithmetic expressions or character expressions. Macro model statements can determine attributes of set symbols or macro arguments. RRMAC allows macros to define new macros and to call other macros. Macro calls can be recursive, that is, a macro may call itself.

RRMAC contains a mini co-resident editor, allows spooling if desired, supports insert files, retains TSC Text Editor source code compatability, plus many other programmer conveniences features.

RRMAC can be furnished on cassette or mini-floppy in either SSB or Mini-Flex format. Comes complete with linking loader, Instruction Manual/Programmers Guide & an extensively commented assembly listing.

M68RR . . . cassette . . . $160.00
M68RR-D . . . SSB disc . . . $160.00
M68RR-F . . . FLEX disc . . . $160.00
Specify "Mini" or Ver. 2.0

A 6800 to 6809 cross assembler version of RRMAC will be available in June.

Order directly by check or MC/Visa. California residents add 6% sales tax. Customers outside of U.S. or Canada add $5 for air postage & handling.

Dealer inquiries welcome.

FLEX is trademark of TSC

Ed Smith's SOFTWARE WORKS
P.O. Box 339, Redondo Beach, CA 90277, (213) 373-3360

ATTENTION SWTPC DISC SYSTEM USERS

All Ed Smith's M6800 SOFTWARE TOOLS

are now available in mini FLEX format as well as Smoke Signal Broadcasting format. See July BYTE for diskette media prices. Disc users specify SSB OR SWTPCO.

M6800 RELOCATING ASSEMBLER AND LINKING LOADER software for rapid program development and debugging. Some of its features are RELOCATABLE CODE-FULL ALPHABETIZED CROSS REFERENCE LISTING-BOTH GLOBAL AND LOCAL LABELS-LISTING OF EXECUTION TIME-RELATIVE BRANCH TARGET ADDRESSES-8 CHARACTER LABELS-CORESIDENT EDITOR-ENGLISH ERROR MESSAGES-60 COLUMN LISTING USING PR40, etc.

M68AS . . cassette $50.00
M68AS-D . . minifloppy $55.95

M6800 RELOCATABLE DISASSEMBLER AND SEGMENTED SOURCE TEXT GENERATOR. This software tool enables you to modify and adapt those large sized object programs and re-assemble them on your system without requiring enormous memory. This program will produce segmented source text files with all the external linkage information required for re-assembly using the above Relocating Assembler.

M68RS . . cassette $35.00
M68RS-D . . minifloppy $40.95

M68ASPK Above two programs on single diskette $85.00

The above two programs are furnished in relocatable formatted code with instruction manuals and commented assembly listings. The loader is also supplied in standard MIKBUG format on cassette or as a binary file on disc to initially load the linking loader at any desired address.

The linking loader is also available in EPROM on two 2708's for $45.00. Specify desired address and version, i.e. cassette, FLEX or SSB. For FLEX Specify "Mini" or Ver 2.0

S-M-I-T-H-B-U-G

FIRMWARE from Ed Smith's SOFTWARE WORKS

A 2k Monitor Disassembler-Trace Debugging Tool in EPROM. Use as stand-alone monitor or as an adjunct to your Smartbug or Swtbug. Requires ACIA as control port. Provides all the monitor commands (23) you could ever ask for, plus the convenience of a mnemonic disassembler and single stepping disassembler trace display. Interactive use of single or dual breakpoint with trace pickup at breakpoint, plus many new monitor functions and subroutines, makes this a super tool you will love to work with. Uses location $F800 to $FFFF. Furnished on two 2708's or one 2716. Includes Manual and commented assembly listing. Specify ACIA location and present monitor in order to keep $A000 RAM compatibility.

SMITHBUG ON 2708's $60.00
SMITHBUG ON 2716 . . . . . $70.00

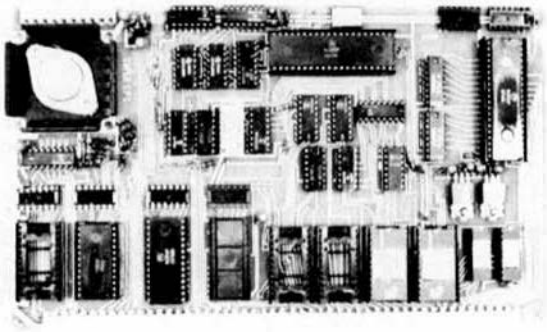
Get under Ed Smith's Software RUG (Relocating assembler Users Group). All RUG members will have use of a building software support library in relocatable format. Upcoming items are a Floating Point package and a Trig package.

Order direct by check. Specify system configuration if other than SwTPCo. California residents add 6% sales tax.

Ed Smith's SOFTWARE WORKS
P.O. Box 339, Redondo Beach, CA 90277, (213) 373-3350

NOW INDUSTRIAL QUALITY AT LOW COST
FROM THOMAS INSTRUMENTATION

Industrial system boards are now available separately for OEM, prototyping or hobbyist applications. Shipped from stock, these are the same quality cards used in monitors and machine tool controls designed for GM, LTI, and General Electric. All cards are SS-50 buss compatible and are suitable for dedicated applications. The CPU card and the Video RAM Card may be combined on a TI backplane as a stand-alone micro — ideal for prototypes or hobbyists. ATTENTION OEM's: If you have a control data acquisition, monitoring, or other microprocessor application. Check with TI for more information about custom software design for the TI CPU or any other 6800 series system. TI also has non-SS-50 buss single board 6800 systems.



SS-50 SUPER CPU

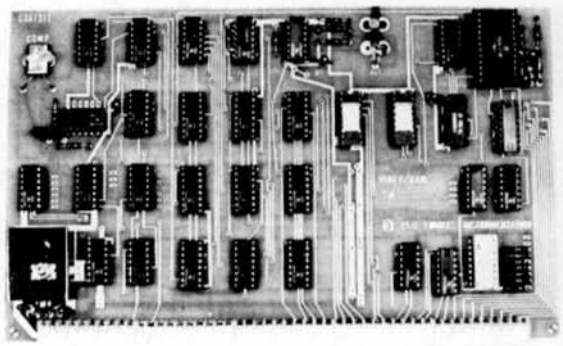
- SS-50 or stand alone computer
- 1K of RAM at $8000. I/O on board at $8400 (Relocatable)
- 2K Monitor (Mikbug compatible) in 2708 EPROM
- 2 8 bit parallel ports with 2 control bits and power
- RS-232 ACIA port. 2nd TTL ACIA optional
- 3 16 bit counter/timers (expandable to 6 add 2nd 6840)
- 128 byte RAM at 0000 is jumper selectable
- Battery back-up for 32 bytes of RAM
- Plug back to back with Video RAM for Stand alone micro or customized smart terminal

ASSEMBLED $195.00
CARD AND DOCUMENTATION $49.00

SS-50 VIDEO RAM

- Fully synchronous operation — No jitter
- 7 by 9 Characters Programmable reverse video
- Full 1 .8 Character ASCII set
- 1K of memory can be mapped to any 1K boundary
- Full documentation includes software (Replaces OUTEE)

ASSEMBLED $149.00
CARD, CRYSTAL and DOCUMENTATION $39.00



TI SS-50 Wire-Wrap Card	**24.00**
TI SS-50 Parallel I/fl flard	**95.00**
Card only	**35.00**
GIMIX I6K Static RAM w/Soft addressing	**359.00**
GIMIX 16 RAM without Soft addressing	**290.00**
3, 4, 7 SLOT Backplanes (per slot)	**4.00**

TI cards available from stock

THOMAS INSTRUMENTATION

2709 Dune Drive, Avalon, N.J. 08202
Phone (609) 967-4280

VISA

DEALERS FOR GIMIX SWTPC SSB

CALL FOR DEALER, OEM. AND QUANTITY PRICES

'68' MICRO JOURNAL

★ The only ALL 6800 Computer Magazine.

★ **More** 6800 material than all the others combined:

MAGAZINE COMPARISON
(2 years)
Monthly Averages
6800 Articles

KB	BYTE	CC	DOBB'S	TOTAL PAGES
7.8	6.4	2.7	2.2	19.1 ea. mo.

Average cost for all four each month: $5.88
(Based on advertised 1-year subscription price)

'68' cost per month: 88¢
($10.50 Charter Subscription Rate)

That's Right! Much, Much More
for
1/6 the Cost!

CHARTER SUBSCRIPTION SPECIAL

1-Year $10.50 2 Years $18.50 3 Years $26.50

OK, PLEASE ENTER MY SUBSCRIPTION

Bill My: Master Charge ☐ — VISA ☐

Card #_____ Exp. Date_____

For ☐ 1-Year ☐ 2 Years ☐ 3 Years

Enclosed: $_____

Name_____

Street_____

City_____State_____Zip_____

My Computer Is:_____

68 MICRO JOURNAL
3018 Hamill Road
HIXSON, TN 37343

Foreign surface add $9.50 per year.
Foreign Air Mail add $29.00 per year.

SUPER SOFTWARE!

MICROWARE 6800 SOFTWARE IS INNOVATION AND PERFORMANCE

NEW LISP Interpreter

The programming language LISP offers exciting new possibilities for microcomputer applications. A highly interactive interpreter that uses list-type data structures which are simultaneously data and executable instructions. LISP features an unusual structured, recursive function-oriented syntax. Widely used for processing, artificial intelligence, education, simulation and computer-aided design. 6800 LISP requires a minimum of 12K RAM.
Price $75.00

A/BASIC Compiler

The ever-growing A/BASIC family is threatening old-fashioned assembly language programming in a big way. This BASIC compiler generates pure, fast, efficient 6800 machine language from easy to write BASIC source programs. Uses ultra-fast integer math, extended string functions, boolean operators and real-time operations. Output is ROMable and RUNS WITHOUT ANY RUN-TIME PACKAGE. Disk versions have disk I/O statements and require 12K memory and host DOS. Cassette version runs in 8K and requires RT/68 operating system.
Price: Disk Extended Version 2.1 $150.00
Cassette Version 1.0 $65.00

NEW A/BASIC Source Generator

An "add-on" option for A/BASIC Compiler disk versions that adds an extra third pass which generates a full assembly-language output listing AND assembly language source file. Uses original BASIC names and inserts BASIC source lines as comments. SSB and SWTPC Miniflex version available.
Price: $50.00

NEW A/BASIC Interpreter

Here it is—a super-fast A/BASIC interpreter that is source-compatible with our A/BASIC compiler! Now you can interactively edit, execute and debug A/BASIC programs with the ease of an interpreter—then compile to super efficient machine language. Also a superb stand-alone applications and control-oriented interpreter. Requires 8K RAM. The cassette version is perfect for Motorola D2 Kits.
Price: $75.00

RT/68 Real Time Operating System

MIKBUG—compatible ROM that combines an improved monitor/debugger with a powerful multitasking real-time operating system. Supports up to 16 concurrent tasks at 8 priority levels plus real time clock and interrupt control. Thousands in use since 1978 handling all types of applications. Available on 6830 (MIKBUG-type) or 2708 (EPROM-type) ROM. Manual is a classic on 6800 real-time applications and contains a full source program listing.
Price: RT68MX (6830) $55.00
RT68MXP (2708) $55.00

6800 CHESS

A challenging chess program for the 6800. Two selectable difficulty levels. Displays formatted chess board on standard terminals. Requires 8K memory. Machine language with A/BASIC source listing.
Price: $50.00

ELIZA

6800 version of the famous MIT artificial intelligence program. The computer assumes the role of a psychoanalyst and you are the patient. This unusual program is unique because the dialog with the computer is in unstructured plain English. An impressive demonstration program.
Price: $30.00

Our software is available for most popular 6800 systems on cassette or diskette unless otherwise noted. Disk versions available on S.S.B., SWTPC, or Motorola MDOS. Please specify which you require. Phone orders are welcomed. We accept MASTERCHARGE and VISA. We try to ship orders within 24 hours of receipt. Please call or write if you require additional information or our free catalog. Microware software is available for OEM and custom applications.

MICROWARE SYSTEMS CORPORATION
P.O. BOX 4865
DES MOINES, IA 50304
(515) 265-6121

We're not just blowing smoke

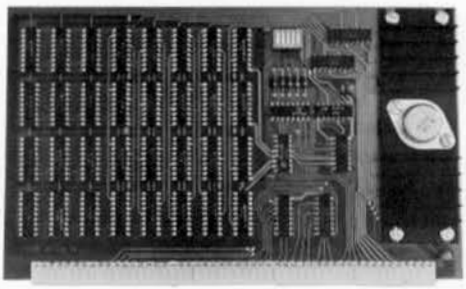
SMOKE SIGNAL BROADCASTING PRESENTS IT'S

$299.00

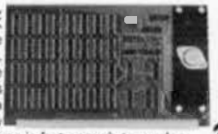
M-16A STATIC MEMORY SYSTEM

• Allows SWTPC 6800 expansion to 48K
• Low Power • Uses Single +8 Volt Supply
• SWTPC 6800 Plug Compatible • STATIC – No refresh required

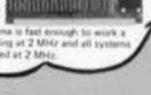
The M-16A STATIC random access memory system, with a total storage capacity of 16834 words of 8 bits each, is switch selectable to any 4K starting address, and a hardware write protect switch is also included. The system's storage elements are 4K by 1 STATIC memory chips which store 4 times as much in only 12% more space than the low power 2102's. Typical access time is fast enough to work a 6800 based computer operating at 2 MHz and all systems are factory tested at 2 MHz.



The M-16A STATIC random access memory system, with a total storage capacity of 16834 words of 8 bits each, is switch selectable to any 4K starting address, and a hardware write protect switch is also included. The system's storage elements are 4K by 1 STATIC memory chips which store 4 times as much in only 12% more space than the low power 2102's. Typical access time is fast enough to work a 6800 based computer operating at 2 MHz and all systems are factory tested at 2 MHz.



The M-16A STATIC random access memory system, with a total storage capacity of 16834 words of 8 bits each, is switch selectable to any 4K starting address, and a hardware write protect switch is also included. The system's storage elements are 4K by 1 STATIC memory chips which store 4 times as much in only 12% more space than the low power 2102's. Typical access time is fast enough to work a 6800 based computer operating at 2 MHz and all systems are factory tested at 2 MHz.



SMOKE SIGNAL BROADCASTING®

31336 Via Colinas, Westlake Village, CA 91361, (213) 889-9340

SMOKE SIGNAL BROADCASTING

31336 Via Colinas, Westlake Village, CA 91361
(213) 889-9340

☐ Send information on your M-16A
☐ Send name of nearest dealer

Name _____
Address _____
Company _____
City _____
State/Zip _____

'68' Micro Journal
3018 Hamill Rd.
Hixson, TN 37343

Second Class Postage
Paid at Chattanooga TN